DEDICATION

I dedicate this work

To My Parents

Late Mr. Razi Ahmad Khan and Mrs. Shakeela Bano
For their love, care and support

ACKNOWLEDGEMENT

عَلَّمَ الإِنْسَانَ مَا لَمْ يَعْلَمْ

"Allamal-insanamalamya'lam"

(Teacheth the man, which he knew not)

Without any doubt, first and foremost, I would like to pay my humble reverence to almighty ALLAH, the most merciful and beneficent, without his blessings the work would not have seen the light of the day.

The completion of a research work has never been a 'one man show' but collective efforts of all the well wishers. This thesis has been very exciting and challenging for me and I have been accompanied by a great number of people whose contributions are worth to be mentioned.

I am very much indebted and wish to express my special appreciation and thanks to my mentors as well as my supervisor Prof. Jamshed Siddiqui, Professor, Department of Computer Science and Co- supervisor Dr. Abdus Samad, Associate Professor, University Women's Polythecnic, Aligarh, Aligarh Muslim University for giving the Midas touch to my thesis. I would like to thank for encouragement and unconditional support which allows me to grow as a researcher. Their long and successful experience in the field of Parallel Processing proved very fruitful to me. Their advice on my research work has been invaluable. They not only taught me to be a good scholar, but to be a good person in the life also.

I express my sincere gratitude to Prof. Mohammad Ubaidullah Bokhari, Chairman, Department of Computer Science, Faculty of Science, Aligarh Muslim University, Aligarh, for providing me all the necessary research facilities in the Department.

I would like to convey my heartfelt thanks to all my teachers, Dr. Rafiqul Zaman Khan, Dr. Tamana Siddiqui, Mr. Arman Rasool Faridi and Mr. Faisal Anwer Department of Computer Science, Aligarh Muslim University, Aligarh.

I acknowledge my deepest gratitude to the staff of University Women's Polythecnic particularly Mrs. Savita Gautam, Assistant Professor & Mr. Javed for their cooperation while working in the lab of Women's Polythecnic.

I have pleasure in expressing my thanks to my uncle Dr. Nabi Ullaha Khan, Associate Professor, Department of Applied Mathematics, A.M.U, Aligarh, for his kind help to provide me right research advice many times whenever I was in need.

I am immensely grateful to all my colleagues specially Mr. Shahab Saquib Sohail, Ms. Suby Khanam, Mr. Nazir Ahmad and Mr. Haider Khalaf Jabbar.

It is delightful for me to thank my senior's Dr. Shadab Alam, Dr. Shamsh Tabrez Siddiqui, Mr. Faraz Hasan for their advices and intial support during my research which is really appreciable work. I cannot forget my affectionable juniors Mr. Riyaz Ahmad, Mr. Ausaf Ahmad, Mr Parvej Aalam, Mr. Oquail Ahmad, Mr Rizwan Aalam, Mr. Shabbir Hasan, Mr. Khashif Adhani and Mr. Mayank Srivastava for their genial relationship. I will always carry the sweet memory of you people with me. You all are more than just a colleague to me.

It will be injustice, if I will not remember my dear friends at this juncture for being with me in all my good and bad times during my long stay at the Aligarh Muslim University. I feel very blessed when I count few names among them like Dr. Yasir Akhtar Khan, Mr. Syed Mohammad Faisal, and Mr. Raffan Ali.

Apart from the above, I would like to extend my heartfelt emotions to my youthful brothers Mr. Haroon Ahmad Khan, Mr. Owais Khan and Mr. Nouman Khan. And also younger cousin sisters Ms Nigar Fatima, Ms. Samreen, Ms Ifza Aziz, Ms Sidra Aziz and Ms Alishba Farhat.

Last but not the least, I would also like to pay my sincere thanks and gratefulness to my mother whose constant encouragement and support acted as an impetus for working hard and completing the work with sincerity. There are no words that can express gratitude for her love, affection and patience. They always stood by my side, have faith in my work and always prayed for my success.

I am deeply grateful to the University Grant Commision, New Delhi, for providing me financial assistance in the form of ***"UGC Non Net"*** *Fellowship during my research.*

Zaki Ahmad Khan

CANDIDATE'S DECLARATION

I, ***ZAKI AHMAD KHAN,*** *Department of Computer Science, certify that the work embodied in this Ph.D. thesis is my own bonafide work carried out by me under the supervision of* ***PROF. JAMSHED SIDDIQUI*** *and* ***DR. ABDUS SAMAD*** *at Aligarh Muslim University, Aligarh. The matter embodied in this Ph.D. thesis has not been submitted for the award of any other degree.*

I declare that I have faithfully acknowledged, given credit to and referred to the research workers wherever their works have been cited in the text and the body of the thesis. I further certify that I have not willfully lifted up some other's work, para, text, data, result, etc. reported in the journals, books, magazines, reports, dissertations, theses, etc., or available at web-sites and included them in this Ph.D. thesis and cited as my own work.

Date:.....

(ZAKI AHMAD KHAN)

CERTIFICATE FROM THE SUPERVISOR

This is to certify that the contents of this thesis entitled ***"PERFORMANCE EVALUATION OF MULTIPROCESSOR SERVER ARCHITECTURES FOR INFORMATION PROCESSING APPLICATIONS"*** *is the original research work of* ***Mr. Zaki Ahmad Khan*** *carried out under my supervision and guidance. He has fulfilled the prescribed conditions given in the ordinance and statutes of Aligarh Muslim University, Aligarh.*

I further certify that the work of this thesis, either partially or fully, has not been submitted to any other University or Institution for the award of any other degree.

Signature of the Supervisor: .....

Name & Designation: Prof. Jamshed Siddiqui

(PROFESSOR)

Department: COMPUTER SCIENCE

Signature of the Co-Supervisor: .....

Name & Designation: Dr. Abdus Samad

(ASSOCIATE PROFESSOR)

UNIVERSITY WOMEN'S POLYTHECNIC

(Signature of the Chairman of the Department with seal)

COPYRIGHT TRANSFER CERTIFICATE

TITLE OF THE THESIS: *PERFORMANCE EVALUATION OF MULTIPROCESSOR SERVER ARCHITECTURES FOR INFORMATION PROCESSING APPLICATIONS*

CANDIDATE'S NAME: ***ZAKI AHMAD KHAN***

COPYRIGHT TRANSFER

The undersigned hereby assigns to the Aligarh Muslim University, Aligarh, copyright that may exist in and for the above thesis submitted for the award of Ph.D. degree.

(ZAKI AHMAD KHAN)

COURSE/ COMPREHENSIVE EXAMINATION/ PRE-SUBMISSION SEMINAR COMPLETION CERTIFICATE

This is to certify that ***Mr. Zaki Ahmad Khan****, Department of Computer Science has satisfactory completed the Course Work/Comprehensive Examination and Pre-submission seminar requirement which is part of his Ph.D. programme.*

Date:..... **(Signature of the Chairman of the Department)**

ABSTRACT

The internet begins to grow quickly in the recent years as a results user dependency as well as expectation enhanced. The increase bandwidth and sophisticated network infrastructure has also increased dramatically to stable various internet application services. One of the important applications is accessing information through internet by the end users. Web services take different approaches toward enhancing their performance in the era of informatics applications. These approaches are either software or hardware based. Each one of the two approaches has its own advantages and limitations. In hardware approach, one of the important issues is how to deal with the design of high performance server to attain the desirable performance. Traditional approaches toward the design of server architecture have targeted static contents that are usually network intensive. However, these methods are not applicable to dynamic contents applications which are more compute intensive. In software approach, the performance of the server can be improved by having effective algorithms to improve the server's performance in terms of minimum access time and efficient resource utilization. In hardware approach, additional computing power can be achieved by applying various techniques together and also by adding more processors in to the single system. With the technological development the cost of installing a multiprocessor system is significantly reduced. Therefore, multiprocessor approach to design and develop a particular architecture is the most generalized and flexible one. The interconnection networks are critical components of this approach in order to efficiently connect different processing elements to retain high performance.

The present work is concerned with the design and development of a new multiprocessor architecture and to use it in a variety of information processing applications. The performance of the proposed architecture is tested for both; handling the unpredictable

communication traffic and servicing a number of information retrieval queries in a variety of mode. A comparative study is made by studying various characteristics of similar multiprocessor architectures. Based on this analysis a new multiprocessor architecture as **"Linear Crossed Cube (LCQ)"** network is proposed. The proposed LCQ is a linear type of architecture which removes the major drawback of exponential expansion in cube based architectures. The LCQ network is less complex by having lesser number of nodes and constant number of links. It inherits most of the desirable topological properties such as smaller diameter, constant node degree and high bisection width. The proposed topology is highly scalable and can be extended to the higher level with virtually no alteration in the existing configuration. The complexity of extension in LCQ increases linearly and each extension at n^{th} level requires n+1 or n+2 nodes. These extension are referred to as add and even extension respectively.

To obtain a high performance of a parallel system, it is essential to distribute the load (task) evenly among different processing elements also known as nodes. The performance of LCQ network is tested by implementing several dynamic scheduling schemes. The dynamic scheduling works on the fly that is all information obtained as scheduling is in progress. A novel dynamic scheduling scheme is proposed named as **"Optimal Multi-Step Scheduling (OMSS)"** to utilize all the nodes of the LCQ network effectively. The performance is evaluated in terms of Load Imbalance Factor (LIF's) i.e. the imbalance of load left after a balancing action and the balancing/service time at various stages of the load. The Proposed Scheme works on the basis of minimum distance property that is it takes care of all the directly connected processors in the network. However, the proposed algorithm is designed to consider in direct connectivity of nodes while processing lesser communication cost and overhead on the scheduler. The proposed scheme is also implemented on other similar multiprocessor network namely Hypercube (HC) and Star Crossed Cube (SCQ). The comparative simulation study shows that the proposed scheme provides more advantageous performance in term of task Scheduling on LCQ network. The other dynamic scheduling schemes such as Minimum Distance Scheduling (MDS) and Two Round Scheduling (TRS) are also implemented and tested on LCQ server. The Proposed Model that is LCQ network with the proposed scheduling scheme shows the superiority of the topology and considered as a better organizational model.

Similarly, another algorithm for information exchange is proposed and implemented on Linear Crossed Cube (LCQ) network to work as server. The algorithm works based on the principal of optimality using analyzer, indexer and query formation. A Table consisting of information of packets and their addresses is maintained which is concurrently accessible by all the nodes of the server. A variety of information retrieval queries are executed and the execution time is evaluated. To prove efficiency the proposed server is tested for different systems in the information retrieval model. These systems are classified on the basis of number of processors used in the server. The performances of these systems are evaluated in terms of different sets of queries namely smaller to medium and medium to larger against execution time. A number of search queries have been examined and the average times taken to process these queries are computed. The simulation results are compared among different systems as well as with Multi-Process uni-processor type system and demonstrated through graphs. It is observed that the LCQ shows a fast retrieval of information with proposed information retrieval algorithm.

The research in this direction indicates that the number of processors in a multiprocessor server architecture are being reduced thereby reducing the cost and complexity of the architecture without losing the performance of the architecture. Therefore, a combination of scalable architecture and efficient algorithm is a better organization model that supports variety of informatics applications.

TABLEOFCONTENTS

LIST OF ABBREVIATIONS

AHS	ADAPTIVE AND HIERARCHICAL TASK SCHEDULING
AMPED	ASYMMETRIC MULTI PROCESS EVENT DIVES
BT	BINARY TREE
CC	CROSSED CUBE
CCC	CUBE CONNECTED CYCLE
COMA	CACHE-ONLY MEMORY ARCHITECTURE
DC	DUAL CUBE
DCPD	DYNAMIC CRITICAL PATH DUPLICATION ALGORITHM
DLH	DOUBLE LOOP HYPERCUBE
DLS	DEPTH LIMITED SEARCH ALGORITHM
FCC	FOLDED CROSSED CUBE
FDC	DUAL CUBE
FMC	FOLDED META CUBE
FPGA	FIELD PROGRAMMING GATE ARRAY
GA	GENETIC ALGORITHM
GM	GRADIENT MODEL
HBM	HIERARCHICAL BALANCING METHOD
HC	HYPERCUBE CUBE
HCN	HIERARCHICAL CUBE NETWORK
IG	ITERATIVE GREEDY ALGORITHM
IL	IDEAL LOAD
IN	INTERCONNECTION NETWORK
IPC	INTER PROCESSOR COMMUNICATION
IR	INFORMATION RETRIEVAL
LA	LINEAR ARRAY
LCS	LOOSELY COUPLED SYSTEMS

LCQ	LINEAR CROSSED CUBE
LEC	LINEARLY EXTENSIBLE CUBE
LET	LINEARLY EXTENSIBLE TREE
LIF	LOAD IMBALANCE FACTOR
MC	META CUBE
MDA	MINIMUM DISTANCE PROPERTY
MDS	MINIMUM DISTANCE PROPERTY
MCP	MODIFIED CRITICAL PATH ALGORITHM
MP	MULTI PROCESS
MPSoC	MULTIPROCESSOR SYSTEM ON CHIP
MT	MULTI THREAD
NH	NECKLACE HYPERCUBE
NOC	NETWORK ON CHIP
NUMA	NON-UNIFORM MEMORY ACCESS
OMSS	OPTIMAL MULTI-STEP SCHEDULING
PD	PARALLEL DOWNLOADING
PE	PROCESSING ELEMENT
R	RING
RHC	REDUCED HYPERCUBE
RIL	ROUNDED IDEAL LOAD
SCQ	STAR CROSSED CUBE
SoC	SYSTEM ON CHIP
SPED	SINGLE PROCESSOR EVENT DRIVEN
TCS	TIGHTLY COUPLED SYSTEMS
TGS	TASK GENERATED AT A PARTICULAR LOAD STAGE
TH	TWISTED HYPERCUBE
TRS	TWO ROUND SCHEDULING
TWA	TREE WALKING ALGORITHM
UMA	UNIFORM MEMORY ACCESS
VLSI	VERY LARGE SCALE INTEGRATION

LIST OF FIGURES

LIST OF TABLES

LIST OF FIGURES

INTRODUCTION

1.1 OVERVIEW

Single processor systems have achieved great speeds and have been used in high performance computing. However, this trend came to an end because there are physical and architectural bounds, particularly the physical limit of chip manufacturing. A parallel system in the form of internally linked processors is an alternative approach to increase computing power. Recently, advances in computer networks have created a new type of parallelism in the form of networked autonomous computers. Instead of putting everything in a single box i.e. tightly couple processors to memory; the Internet achieved a kind of parallelism by loosely connecting everything outside of the box [Gordon et al., 2006]. To obtain the best possible use of a high performance computer system with internal or external parallelism, it is necessary to understand the interaction between hardware and software parts of the system [Pancake et al., 1996]. Today numerous applications require more computing power than a sequential computer can offer. Parallel processing provides a cost effective solution to this problem by increasing the number of CPUs also known as Processing Elements (PEs) in a computer and by adding an efficient communication system between them. The work-load can now be shared between these connected processors. These interconnected processors results in much higher computing power and performance that could not be achieved with traditional single processor system. There are many applications of parallel processing. The most notable are VLSI design, engineering analysis (CAD/CAM), military applications and global climate modelling and forecasting etc. The parallel processing environment has the potential for providing significant computing power at a fraction of the cost of fourth-generation supercomputers [Balram et al., 1988], [Tiwari et al., 2015]. The technology of parallel

processing is mature and can be exploited commercially. There is already significant research carried out on the development of tools and environment related to parallel processing.

Internet is described as the worldwide publicly accessible network of interconnected computer networks that transmit data by packet switching using the standard Internet Protocol TCP/IP. It also interacts as an economical interaction platform for research and development, education and amusement and almost in all the perspective. Accordingly, the population of Internet users is boosting quickly over the past couple of years. This massive user base is putting considerable pressure on the computing assets of well-liked services accessible online. Given that, useful and also dependable information retrieval via internet is now more and more crucial. That is especially needed in multimedia applications particularly when streaming of audio and video information is performed. The achievements of the Internet broadside from the features to assistance efficient, survivable, robust and reliable end-to-end data transfer services for flexible applications functioning over a couple of end-systems. Popular documents preserved at a server can attract enormous access requests which cause an immoderate rise in client requests. However, user's expectation has improved to ensure that the desired information ought to be downloaded in the least possible time [Gkantsidis et al., 2003]. As a result, the server plays an important role to manage the load and should not be treated as bottleneck. Site managers regularly deal with the need to improve server capacity. In lieu of finding the best server to consummate a client's demand and concurrent use of a couple of servers for satisfying demand additionally kenned as Parallel Downloading [Philopoulos and Maheswaran, 2001], [Nor et al., 2013]. This has been implemented by numerous Internet file downloading applications. The mimic of parallel downloading is to increase the download speed as compared with a single link from a single server. Within this method, a client requesting a file will open concurrent connections to multiple senders which are often servers or peers [Koo et al., 2003]. To decrement the load on the unspoiled file server and mirror servers tend to be used. Consequently, various aspects of the file will be transmitted from the sender to the client. There have been tests displaying that parallel downloading results in high throughput, thereby obtaining smaller downloading time by the end users [Jin et al., 2012]. The shorter downloading time, nevertheless is received at

the expenditures of much more overheads on handling the file to be downloaded from several servers as well as by incorporating sustaining management among servers [Haitao et al., 2005]. Another approach to handle the problem of information exchange on the network is to design high performance architecture that incorporate the concept of multiprocessor architectures [Foglia et al., 2000]. These servers can utilize the multiprocessing capability having n-processors under a single domain and hence provide cost-effective solution. This strategy meritoriously has used profitable servers such as e-Bay and Google.

1.2 OVERVIEW OF HIGH PERFORMANCE COMPUTING

In the research area, traditional High performance computing (HPC) refers to use of supercomputing, grid environments and/or clusters of computers to solve computation intensive problems. The common aim is to accelerate data parallel computations in large scale parallel applications. Many supercomputers in the TOP500 list contain hundreds of thousands of computing nodes for incorporating data parallelism. For instance Titan, a Cray XK7 system contains 560, 640 processors [Karunadasa and Ranasinghe, 2009], [Top500 et al., 2013]. These huge numbers of nodes are to be connected in order to retain high performance with the physical limitations of currently available hardware. The Grid computing enables the sharing, selection and aggregation of a wide variety of geographically distributed resources including supercomputers, storage systems, data sources and specialized devices owned by different organizations for solving largescale resource-intensive problems in science, engineering and commerce [Baker et al., 2002], [Venugopal et al., 2006]. Cluster computing which consists of a collection of inter-connected stand-alone computers working together as a single integrated computing resource [Holmes and Kureshi, 2015]. Cloud computing is the most recently reported framework that incorporates characteristics of both clusters and grids. It consist of a collection of inter-connected and virtualized computers that are dynamically provisioned and presented as one or more unified computing resources based on service-level agreements established through negotiation between the service provider and consumers [Deelman et al., 2008], [Buyya et al., 2009], [Tchana et al., 2013].

1.3 DESIGN ISSUES OF SERVER

The design of information retrieval systems must address not only issues of look-and-feel but also of effective interaction. Dialog models based on relevance feedback and query reformulation explicitly address the ill-defined nature of information seeking by allowing users to learn from the repository and iteratively refine the information need. Systems need to support a number of interaction styles such as querying and browsing to accommodate the different kinds of search strategies users may need to use [Mishra and Mishra, 2008].

Design strategies for retrieval systems need to pay particular attention how fast user's access information from the hardware. Therefore, effectiveness of information retrieval algorithm is also dependent on the physical network topology. Moreover, if they are bottlenecks in different network connection the performance is not increased and these additional connections simply waste the server resources. To overcome these bottlenecks enhancing the server performance has become the critical issue during the design and development of such systems [Hellerstein et al., 2007]. Besides, achieving additional computing power by incorporating more than one processing element, the performance of the server can be enhanced by running parallel algorithms for accessing information. Information analysis, user modelling and interaction modelling are some other methods that can be incorporated to improve the effectiveness of algorithm. A combination of parallel hardware with efficient software approach optimizes the system performance.

1.4 PERFORMANCE ISSUES OF SERVER

Improving the performance of servers has become a critical issue to cope with the increasing use of network based services. Due to increase in the demand of information transactions and scheduling overhead the existing approaches have some limitations. The critical nature of many online transactions require a high performance server since such servers are anticipated to be the bottleneck in hosting Internet based services. Multiprocessor approach is the most powerful and economical way to achieve desirable performance by incorporating efficient network topologies. Numerous interconnection topologies have been designed to achieve the desired performance. Nevertheless, the actual performance is far below the expectation of users when executing parallel

applications on a particular multiprocessor network. The performance of multiprocessor interconnection topologies can be evaluated in terms of numerous topological parameters along with the routing techniques involved. These parameters along with suitable routing can increase the performance of multiprocessor architectures [Feng et al., 1981], [Adhikari and Tripathy, 2011]. The load balancing is also an important requirement which has a scope to improve the overall performance. Load balancing parameters namely Load Imbalance Factor (LIF's) as well as execution time are critical parameters for achieving high performance. The aim is to design and structured approach for running massively parallel application with multiple processing elements at backend. Therefore, efficient scheduling algorithms are designed that partitioned the task and schedule them onto the physical processors. Similarly, to manage the traffic a dynamically optimal solution is essential by means of programming models [Gopal et al., 1996] [Galindo et al., 2008] [Haroon and Hussain, 2013]. In the present work, an idea is presented to overcome some of the problems of delay in accessing and downloading the information using multiprocessing technique. A new scalable network topology called Linear Crossed Cube (LCQ) multiprocessor is proposed to be used as server. Through simulation the performance of the proposed server is evaluated for both the load balancing and exchanging information simultaneously.

1.5 MOTIVATION

The main idea of the research is to design a multiprocessor interconnection network with lesser number of nodes having better characteristics than the similar existing networks. The lesser number of nodes means economical. The other important characteristics of multiprocessor network are diameter, connectivity, extensibility and fault tolerance etc. The success of a multiprocessor system depends upon the effective utilization of nodes with uniform load distribution. The load distribution is optimal when all the nodes share equal amount of load. The present work is motivated by the requirement for the design and development of a network model to achieve higher performance by comparing the different network characteristics with other networks. To evaluate the network utilization, a number of dynamic scheduling schemes will be implemented on the proposed

architecture. The performance parameters will be calculated and will be compared with other similar multiprocessor networks.

The design of a low cost multiprocessor architecture with the above objectives provides much in the way of designing and implementing highly efficient multiprocessor server. Using the LCQ server, the downloading mechanism continues to offer significant performance improvements compared to single node server architecture for documents of several hundred or thousands of kilobytes. The implementation of the proposed server can be easily integrated in the present web services.

1.6 ORIGINAL CONTRIBUTION

The complete work as presented in this thesis can be divided into three parts. The first part is concerned with the design and development of a low cost multiprocessor architecture. A new multiprocessor topology called Linear Crossed Cube (LCQ) network is proposed. It includes the advantages of hypercube topology such as symmetry, smaller diameter, good connectivity and high bisection width. It is a highly scalable architecture that consists of constant node degree independent to the network size.

The thesis also proposed a scheduling scheme that can maintain the efficient utilization of all the processing elements available in the network for various types of load. To minimize the balancing time, a novel dynamic scheduling scheme named as Optimal Multi-Step Scheduling (OMSS) scheme has been proposed and implemented on LCQ network. The OMSS scheme schedules the tasks on LCQ network whereas the load is divided into approximately equal number of packets to achieve load balancing in the system. The OMSS scheme is also implemented on other standard reported multiprocessors architectures and a comparative study is carried out which shows a minimum balancing time on the LCQ network.

The third and last part of the thesis presents the implementation of a novel information retrieval algorithm which manages the exchange of information on LCQ network. A number of search queries have been examined and the average times taken to process these queries are computed. These time estimates are compared with time taken by several architectures having different set of processors. The simulation results show that the given multiprocessor architecture when implemented as a server with the

proposed algorithms reduces the resource download time.

1.7 THESIS OUTLINE

The rest of the thesis is organized into seven chapters. A chapter wise outline of the thesis is presented below:

Chapter 2: Review of Multiprocessor Interconnection Networks: In this chapter the basic concepts and properties of various multiprocessor networks have been discussed. The chapter describes in detail a review of various multiprocessor architectures reported in the literature. Their characteristics and performance parameters are presented. The various limitations and shortcomings of the available multiprocessor architectures have been mentioned.

Chapter 3: Review of Scheduling Algorithms: In this chapter a review of the scheduling algorithms starting from the classification to the present scenario is presented. A comparative discussion is made and the various factors influencing the run time overheads are discussed.

Chapter 4: Linear Crossed Cube Network: In the light of study and survey performed for various multiprocessor interconnection architectures, this chapter describes in detail the design of the proposed multiprocessor architecture (i.e. LCQ) and its characteristics. To evaluate the performance of the proposed LCQ network a comparative study is carried out in terms of various topological parameters.

Chapter 5: Performance Measure Strategies: This chapter deals with the scheduling of load on the proposed architecture. A new dynamic scheduling scheme named as Optimal Multi-Step Scheduling (OMSS) scheme has been proposed and described. The OMSS scheme has been implemented on LCQ and other standard reported multiprocessor architectures and the simulation results are obtained. In order to evaluate the performance of the proposed OMSS scheme, other standard dynamic schemes are also implemented on LCQ and the simulation results are discussed.

Chapter 6: LCQ as Information Retrieval Server: This chapter is devoted for loading and retrieval of information on the LCQ. The proposed LCQ has been tested to work as a server. An information retrieval algorithm is proposed and implemented for exchange of

information from the LCQ. Through simulation study the relative performance of the proposed algorithm is evaluated by executing different set of queries.

Chapter 7: Conclusion and Future Work: It concludes the overall work and emphasizes the positive points of the proposed organizational model along with the scope for future extension of the work.

REVIEW OF MULTIPROCESSORS INTERCONNECTION NETWORKS

2.1 OVERVIEW

Parallel computing has been widely studied in the field of computer science and has proven to be critical when researching high performance solutions. Modern computer systems employ multiple processors which may take part in execution in parallel to enhance the performance of an application. The benefits of parallel computing need to take into consideration the number of processors being deployed, the complexity of the system as well as the communication incurred between various processing units known as nodes. The topology of interconnection networks plays a key role in the performance of parallel computing systems. There are two types of parallel system such as shared memory system and message passing system. Shared memory systems are a major part of parallel system. A shared memory computer system consists of a set of independent processors, a set of memory and an interconnection network. The whole interprocessor co-ordination and synchronization is accomplished through the global memory. It is also known as tightly coupled systems [Grama et al., 2003], [Jordan and Alaghband, 2003]. The communications among tasks running on different processors are performed by reading and writing from the global memory that is equally accessible by all processor. The shared memory systems can be classified: Uniform Memory Access (UMA), Non-Uniform Memory Access (NUMA) and Cache-Only Memory Architecture (COMA) [Sorin et al., 2003]. In UMA system, a shared memory architecture used in parallel system. The whole processors share the physical

memory uniformly through an interconnection network in the same way a single processor access its memory. All the processors have independent access time to a memory location of which memory chip contains the transferred data. The UMA model is suitable for general purpose and time sharing applications by multiple users. The UMA systems have few drawbacks of memory bottleneck and scalability [Kandemir and Choudhary, 2002], [Shivakumar and Jouppi, 2001]. A memory bottleneck is a stage in a process that causes the entire process to slow down or stop [Anderson et al., 2003]. In a communications context, a bottleneck is a point in the enterprise where the flow of data is impaired or stopped entirely. To remove these drawbacks, NUMA systems are designed. In these systems, system memory design used in multiprocessing where the system memory access time depends on the memory location relative to the processor. A processor can access its own local memory faster than non-local memory under NUMA. The benefits of NUMA are limited to particular workloads, notably on servers where the data are often associated strongly with certain tasks or users. Similar to the NUMA, each processor has part of the shared memory in COMA. However, in this case the shared memory consists of cache memory. A COMA system requires that data be migrated to the processor requesting it [Joseph et al., 2006], [Kwak and Jhon, 2007], [Tanveer et al., 2011], [Tripathy and Tripathy, 2011].

The Communication of data among a set of processors without the need for global memory is incorporated in message passing system. In these systems, each processor has its own local memory and communicates with other processors using messages. All the processing is done locally. There is no sharing of address space. Message passing differs from conventional programming where a process, subroutine, or function is directly invoked by name. Message passing is a key to some models of concurrency and object-oriented programming. Unlike shared memory systems, communication in message passing systems are performed via send and receive operations. These systems are also referred to as a Loosely Coupled Multiprocessor Systems (LCS) [Jung et al., 2006], [Lee et al., 2009]. Distributed systems are built easily using this type of architecture. Besides being cheap, scalability is one of its major properties. It may be increased as and when needed. For truly distributed system, loosely coupled systems are more suitable because they offer scalability at an economical price. Significant progress has been made in the past decades in the design and development of efficient interconnection networks [Chhabra and Sing, 2009], [Parhami

et al., 2000] [Kim and Veidenbaum, 1999], [Cheng and Chuang, 1994], [Shi and Srimani, 2005], [Saad and Schultz, 1998]. Hypercube architecture is a promising approach to improve the performance of multiprocessor parallel systems. Numerous variants of hypercube have been reported in the literature [Amway and Latifi, 1991], [Kumar and Patnaik, 1992], [Efe at al., 1991], [Loh et al., 2005], [Zhang et al., 2002]. These variants include many attractive properties, including regularity, symmetry, small diameter, high connectivity and scalability. A number of approaches have been suggested and implemented to improve the overall performance of such systems [Efe et al., 1992], [Ghose and Desai, 1995], [Kumar et al, 2012], [Khan et al., 2013].

In general interconnection multiprocessor networks are classified in two broad categories based on their topological properties. These are given below:

(1) Cube based network

(2) Linearly Extensible Network

2.2 CUBE BASED NETWORK

The cube based architectures are widely used networks in parallel systems. They have good topological properties such as symmetry, scalability and possess a rich interconnection topology. The Binary hypercube or n-cube has been one of the famous interconnection networks used in the design of parallel systems [Saad and Schultz, 1998]. Besides having a number of desirable features, the major drawback of hypercube is the difficulty of its VLSI layout. The minimum number of tracks for VLSI layout of an n-cube hypercube a one dimensional implementation has an order of network size which results more difficulties to design and fabricate the nodes of the hypercube [Patel et al., 2000], [Patel et al., 2006]. Several variations of hypercube architecture are designed to improve it further. Some examples of hypercube variants are the Folded Hypercube (FH) [Amway and Latifi, 1991], Dual Cube (DC), Folded Dual Cube (FDC) [Adhikari and Tripathy, 2008], Meta Cube (MC) [Peng and Chu, 2002], Folded Met cube (FMC) [Adhikari and Tripathy, 2009], Crossed Cube (CC) [Efe et al., 1992], Folded Cross Cube (FCC) [Adhikari and Tripathy, 2010] and Star Crossed Cube (SCQ) [Adhikari and Tripathy, 2014]. Similarly, modified hypercube architecture named the necklace hypercube which has a good scalability and efficient VLSI layout has been reported [Monemizadeh and Azad, 2005]. These entire variant reducing the

drawback of hypercube and also we can reduce the complexity of the hypercube with the less number of links.

2.3 LINEARLY EXTENSIBLE NETWORK

The Linearly Extensible Networks is another class of multiprocessor architectures which reduces some of the drawbacks of HC architectures. Several researches have been carried out in the design of linear type architectures which are considered less complex and easily extensible. Some examples are Linearly Extensible Tree (LET) [Rafiq et al., 1999], Linear Array (LA), Ring, Star Graph [Altel, Hatel and Krishnamaury, 1987], [Tripathy et al., 2004] and Torus Ring Network [Kwak and Jhon, 2007] etc. The complexity of these networks is lesser as they do not have exponential expansion. Besides the scalability, other parameters to evaluate the performance of such networks are degree, number of nodes, diameter, bisection width and fault tolerance. Selection of a better interconnection network may have several applications with lesser complexities and improved power-efficiency. One such modern application is network on chip (NoC) paradigm where different cores are embedded with appropriate connectivity. Some examples may include mesh, torus, star, etc. [Patel et al., 2011]. Besides their lesser complexity linear architectures severally suffer from the drawbacks of high connectivity and symmetry.

Another class of multiprocessor architecture is hybrid architecture which expects the desirable properties of cube based architecture as well as they are less complex and easily extensible. Thus, they embed the features of various types of architectures in themselves. One such example is Linearly Extensible Cube (LEC) architecture [Samad et al., 2010]. It embedded the desirable properties of hypercube architecture and also has a rich connectivity and symmetry when extended in to higher level. Though the extensibility of LEC is linear still it has a lower bisection width when expanded to higher level of architecture and could not be considered as a good candidate with high fault rate [Zaki et al., 2013].

In the next section the basic concepts of multiprocessor architectures are described followed by an overview of the different topologies used for interconnecting multiple processors with their important parameters are discussed.

2.4 DESIGN ISSUES OF INTERCONNECTION NETWORKS (INS)

The important issue in the design of multiprocessor systems is how to cope with the problem of an adequate design of the interconnection network in order to achieve the desired performance at low cost. The choice of the interconnection network may affect several characteristics of the system such as node complexity, scalability and cost etc. The following are the issues which should be considered while designing an interconnection network.

2.4.1 DIMENSION AND SIZE OF NETWORK

It should be decided how many processing element (PE's) are there in the network and what the dimensionality of the network is i.e. with how many neighbors, each processor is connected.

2.4.2 SYMMETRY OF THE NETWORK

It is important to consider whether the network is symmetric or not i.e., whether all processors are connected with same number of processing elements or the processing elements of corners or edges have different number of adjacent elements.

2.4.3 MESSAGE SIZE

What is a message size? How much data a processor can send in one time unit?

2.4.4 DATA TRANSFER TIME

How long does it take for a message to reach to another processor? Whether this time is a function of link distance between two processors or it depends upon the number of nodes coming in between.

2.4.5 STARTUP TIME

What is the time required to initiate the communication process?

2.5 PERFORMANCE PARAMETERS OF INTERCONNECTION NETWORKS

This section defines the various methods of connecting processors in a parallel computer. A processor organization can be represented by a graph in which the nodes (vertices) represent processors and the edges represent communication channels between pairs of processors.

These processors organization could be evaluated based on certain criteria's or properties of the organization [Quinn et al., 2002], [Hwang et al., 2001], [Mohanty et al., 2008]. These properties help to understand the effectiveness of a particular organization. The various properties are given below:

2.5.1 NUMBER OF NODES (N)

The number of nodes in a multiprocessor network plays a dynamic role by virtue of which the performance of the system is evaluated. Higher number of nodes means higher complexity but higher is the system performance. Therefore, number of processors should be optimal.

2.5.2 NODE DEGREE (D)

The node degree of the network is defined as the number of edges connected with the nodes. It is the connectivity among different nodes in a network. The connectivity of the nodes determines the complexity of the network. The greater number of links in the network means greater is the complexity. If the edge carries data from the node, it is called out degree and if this carries data into the node then it is called in degree.

2.5.3 DIAMETER (D)

The network diameter is defined as the maximum shortest path between the source and destination node. The path length is measured by the number of links traversed. This virtue is important in determining the distance involved in communication and hence the performance of parallel systems. The low diameter is always better because the diameter puts a lower bound on the complexity of parallel algorithms requiring communication between arbitrary pairs of nodes.

2.5.4 COST (C)

It is defined as the product of the diameter and the degree of the node for a symmetric network.

$$\text{Cost (C)} = \text{Diameter} * \text{Degree}$$
$$= D * d$$

Greater number of nodes means greater the cost of the network. It is good creation to measure the hardware cost and the performance of the multiprocessor network and gives more insight to design a cost-effective parallel system.

2.5.5 EXTENSIBILITY

It is virtue which facilitates large sized system out of small ones with minimum changes in the configuration of the nodes. It is the smallest increment by which the system can be expanded in a useful way. A network with large number of links or a large node degree tends to increase the hardware cost. Expandability is an important parameter to evaluate the performance of a multiprocessor system. The feasibility to extend a system while retaining its topological characteristics enables to design large scale parallel systems [Lui et al., 2004].

With this background some important processor organizations have been discussed in the next section.

2.6 REVIEW OF CUBE BASED MULTIPROCESSOR INTERCONNECTION NETWORKS

A number of regular interconnection patterns have evolved over the years. This section presents some of the important multiprocessor interconnection networks reported in the literature and depicted in the Table 2.1. These patterns include:

2.6.1 HYPERCUBE (HC)

Hypercube also known as n-cube or simply binary n-cube is a loosely coupled parallel multiprocessor based on the binary n-cube network. An n-dimensional hypercube contains 2^n nodes and has n edges per node. The Hypercube has been one of the most popular interconnection networks for parallel computer systems. In hypercube, the number of communication links for each node is a logarithmic function of the total number of nodes. In general, an n-cube consists of $N = 2^n$ nodes spanning along n dimensions, with two nodes per dimension. A 3-cube with 8 nodes is shown in Figure 2.1. A 4-cube can be formed by interconnecting the corresponding nodes of two 3-cubes. The node degree of an n-cube equals n and so does the network diameter. The bisection width of that size network is 2^{n-1}. The hypercube organization has low diameter and high bisection width at the expense of the number of edges per node and the length of the longest edge. The length of the longest edge

in a hypercube network increases as the number of nodes in the network increases. In fact the node degree increases exponentially with respect to the dimension, making it difficult to consider the hypercube a scalable architecture. The major drawback of the hypercube is the increase in the number of communication links for each node with the increase in the total number of nodes [Saad and Schultz, 1998], [Zhang, 2002], [Khan et al., 2013]. A number of variants of hypercube are reported in the literatures which are designed to improve the specific topological property.

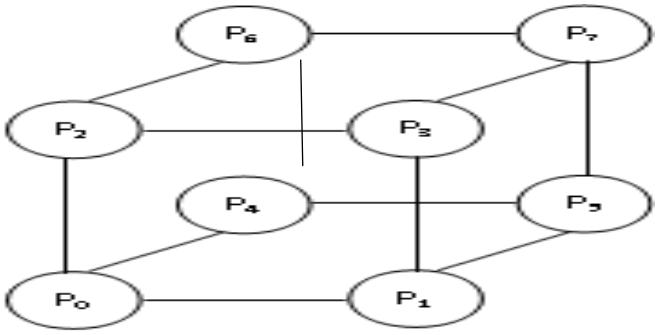


Figure 2.1: An 8-Processor hypercube

2.6.2 CUBE CONNECTED CYCLE (CCC)

The CCC architecture is an attractive parallel computation network suitable for VLSI implementation while preserving all the desired features of hypercube [Parhami et al., 2000], [Khan et al., 2013]. The CCC is constructed from the n- dimensional hypercube by replacing each node in hypercube with a ring containing n node. Each node in a ring then connects to a distinct node to one of the n dimensions. The advantage of the cube- connected cycles is that node's degree is always 3, independent of the value of n. This architecture is modified from hypercube i.e. a 3-cube is modified to form a 3-cube-connected cycles (CCC) restricted the node degree to 3 [Chabbra and Singh, 2009]. The idea is to replace the corner nodes (vertices) of the 3-cube with a ring of 3-nodes. In general one can construct k-cube-connected cycles from a k-cube with $n=2^k$ rings nodes. The idea is to replace each vertex of the k-dimensional hypercube by a ring of k nodes. A k-cube can be thus transformed to a k-CCC with $k \times 2^k$ nodes. The major improvement of a CCC lies in its constant node degree of k,

which is independent of the dimension of the underlying hypercube. On the other hand, it gives a large diameter which ultimately results complex routing then the hypercube [Youyao et al., 2008].

2.6.3 FOLDED HYPERCUBE (FHC)

The FHC is the variation of the hypercube network and constructed by introducing some extra links to the hypercube. Halved diameter, better average distance, shorter delay in communication links, less message traffic density, lower cost make it very promising. The hardware overhead is almost 1/n, n being the dimensionality of the hypercube, which is negligible for large n, optimal routing algorithms are developed are developed and proven to be remarkably more efficient than those of the conventional n-cube. A folded hypercube of dimension n is called FHC (n). The FHC (n) is a regular network of node connectivity (n+1) and the hypercube of degree 3 is converted to FHC (n) network as shown in Figure 2.2. The diameter of an FHC (n) is $\lceil n/2 \rceil$ and bisection width is $2^n/4$ [Amway and Latifi, 1991]. One of the popular extended versions of FHC (n) is Extended Folded Cube (EFC). The EFC has better properties than the other variations of basic hypercube in terms of parameter. It has constant node degree, smaller diameter, and lower cost and also it maintains several numerous desirable characteristics including symmetry, hierarchical, expansive, recursive.

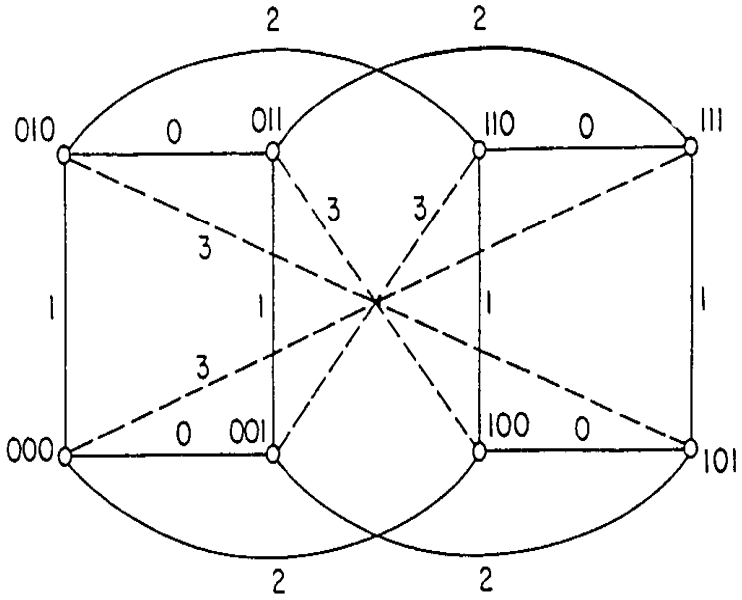


Figure. 2.2: Folded Hypercube

2.6.4 CROSSED CUBE (CC)

The Crossed Cube (CC) has the same node and link complexity as the hypercube and has most of its desirable properties including regularity, recursive structure, partition ability, strong connectivity and ability to simulate other architectures. Its diameter is only half of the diameter of the hypercube. Mean distance between vertices is smaller and it can simulate a hypercube through dilation 2 embedding. The basic properties of the CC, optimal routing and broadcasting algorithms are developed. The CC is derived from a hypercube by Changing the way of connection of some hypercube links. The diameter of CC is almost half of that of its corresponding hypercube [Efe et al., 1992]. The n-dimensional cross cube denoted by CC_n, is the labelled graph defined inductively as follows. CC1 is K2, the complete graph on two vertices with labels 0 and1 as show in Figure 2.3. Specially, the diameter of an n-dimensional Cross Cube is $\lceil (n+1)/2 \rceil$ and the diameter of an n- dimensional hypercube is n. However, the CC makes no improvement in the hardware cost compared to the hypercube. The node connectivity of CC is n with 2^n and it has a bisection width 2^{n-1}. Some variation of cross cube is also available such as Extended Crossed Cube [Adhikari and Tripathy, 2009], Folded Cross Cube, Dynamic Cube and Exchanged Cross Cube [Tripathy et al., 2004], [Adhikari and Tripathy, 2010].

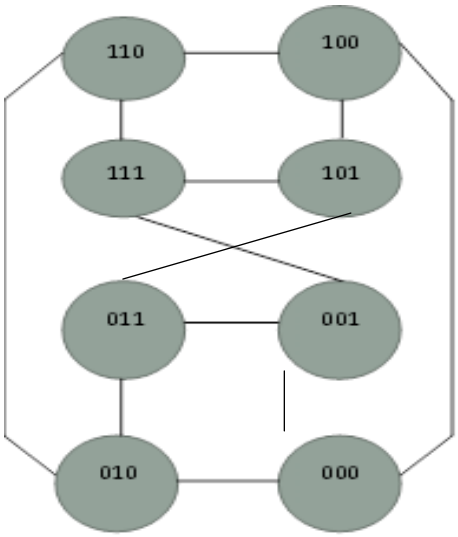


Figure. 2.3: The Crossed Cube (CC3)

2.6.5 REDUCED HYPERCUBE (RHC)

The RH (k, m) is obtained from the n- dimensional hypercube by reducing node edges in hypercube by following rules where k+2m= n. The lower VLSI complexity of RH's permit the construction of systems with more processing elements than are found in conventional hypercube. There are clusters and each cluster is a conventional k- dimensional hypercube. Of the higher n-k=2m dimensions, a node has only one direct connection is decided by the leftmost m bits in the k-bits field, i.e., the (2i + k) dimension, where i is the value of the m-bit binary number. Figure 2.4 shows a reduced hypercube with 3 edges. The number of edges of RH (k, m) is reduced to k+1 [Ziavras et al., 1994].

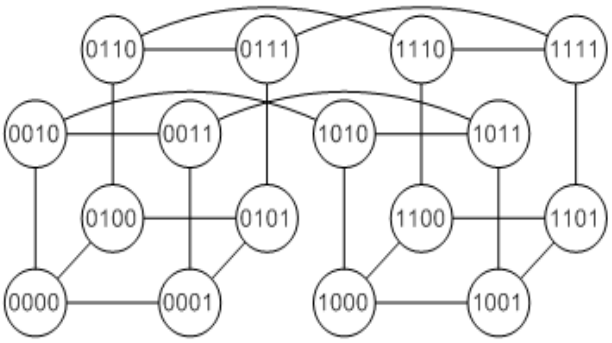


Figure. 2.4: Reduced Hypercube

2.6.6 HIERARCHAL CUBE NETWORK (HCN)

The Hierarchical Cube Network (HCN) is interconnection network for large-scale distributed memory multiprocessors. HCN has about three-fourths the diameter of a comparable hypercube, although it uses about half as many links per node-a fact that has positive ramifications on the implementation of HCN-connected systems. The HCN (n, n) has 2n clusters, where each cluster is an n-cube. Each node in the HCN (n, n) has n+1 links connected to it. Of these, n links are used inside the cluster. The additional links are used to connect nodes among clusters. Figure 2.5 shows a Hierarchical Cube Network with 3 edges. The advantage of HCN is that the number of links required is reduced approximately to half as many links per node and the diameter is reduced to about three-fourth of a corresponding hypercube [Ghosh and Desai, 1995].

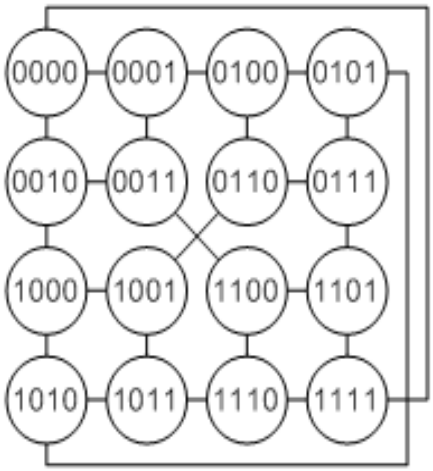


Figure. 2.5: Hierarchical Cube Network

2.6.7 DUAL CUBE (DC)

The DC is a new interconnection topology for large-scale distributed memory multiprocessors. This topology mitigates the problem of increasing number of links in the large-scale hypercube network while keeping most of the topological properties of the hypercube network [Li and Peng, 2000]. The DC shares the desired properties of the hypercube, however increases tremendously the total number of nodes in the system with limited links per node. The key properties of hypercube are also true in the dual-cube: each node can be represented by unique binary number such that two nodes are connected by an edge if and only if the two binary numbers differ in one bit only. However, the size of the dual-cube can be as large as eight thousands with up to eight links per node. A dual-cube uses binary hypercube as basic components. Each such hypercube component is referred to as a cluster. Assume that the number of nodes in a cluster is 2^m. In a dual cube, there are two classes with each class consisting of 2^m clusters. The total number of nodes is $2^m * 2^m$ or 2^{2m+1}. Therefore, the nodes address has 2m+1 bits. The leftmost bit is used to indicate the type of the class (class 0 and class 1). For the class 0, the rightmost m bits are used as the node ID within the cluster. Each node in cluster of class 0 has one and only one extra connection to a node in a cluster of class 1. These two node addresses differ only in the leftmost bit position. The diameter of Gr (V, E) is n+1 and bisection width of DC is 2^{n-2}.

2.6.8 META CUBE (MC)

The MC is an interconnection topology for a very large parallel system. Meta cube network has two level cube structures. An MC (k, m) network can connect 2k+m2k nodes with (k+m) links per node [Peng and Chu, 2002], where k is the dimension of the high-level cubes (classes) and m is the dimension of the low-level cubes (clusters). In this network, the number of nodes is much larger than the hypercube with a small number of links per node. An MC network is a symmetric network with short diameter, easy and efficient routing similar to that of the hypercube. The degree is m+k= (1+n) and the bisection width of an MC (k, m) is $2^{2n}/2$ [Adhikari and Tripathy, 2009]. The met cube has tremendous potential to be used as an interconnection network for very large scale parallel computers since the meta cube can connect hundreds of millions nodes with up to six links per node and it keeps some desired properties of the hypercube that are useful efficient communication among the nodes.

2.6.9 FOLDED DUAL CUBE (FDC)

The FDC is a new cube based Interconnection topology for parallel systems with reduced diameter, cost and constructed from DC and FHC [Adhikari and Tripathy, 2008]. The FDC is a graph Fr (V, E), where V represents a set of vertices and E represent a set of links as shown in Figure 5. The FDC is to be slightly greater than Dualcube but quite less than HC and FHC. Diameter of FDC is found to be smaller than that of Dualcube and with the comparison of Dualcube, HC and FHC. FDC exhibits quite a good improvement in broadcast time over its parent networks with millions of nodes [Khan, Siddiqui and samad, 2015]. The cost of the FDC topology is found to be less. The FDC will help to speed up the overall operation of large scale parallel systems.

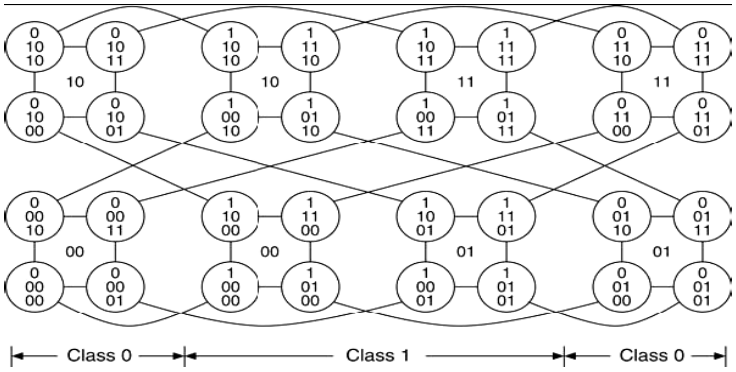


Figure. 2.6: Folded Dual Cube

2.6.10 FOLDED METACUBE (FMC)

The FMC is an efficient large scale parallel interconnection topology with better features such as reduced diameter, cost, improved broadcast time and constructed from MC [Adhikari and Tripathy, 2009]. The FMC is a graph G (V, E) as shown in Figure 6, where V represents a set of vertices and E represent a set of links. The FMC is to be slightly greater than metacube but quite less than HC and FHC. Diameter of FMC is found to be smaller than that of Metacube. The graph is modified of Meta cube. The diameter of FMC is 2n-1 and the bisection width of G (V, E) is $2^{2n}/2 + 2^{2n}$+n-2 [Adhikari and Tripathy, 2009]. The broadcast time of FMC is compared with metacube, FMC exhibits quite a good improvement in broadcast time over its parent network while connecting millions of nodes. The cost of the FMC is found to be less and will help to speed the overall operation of large scale parallel systems.

2.6.11 NECKLACE HYPERCUBE (NH)

This is a new network topology based on the hypercube with an array of processors (as a necklace of processors) attached to each two adjacent nodes of the hypercube network. It is highly scalable architecture while preserving most of the desirable properties of hypercube such as logarithmic diameter, fault tolerance etc. It has also some other properties such as hardware scalability and efficient VLSI layout that make it more attractive than an equivalent hypercube network. The Necklace-Hypercube is an undirected graph which has a necklace of processors to each edge of hypercube. Besides connecting adjacent nodes, the i^{th}-dimension neighbors are connected by an array of nodes as a necklace [Monemizadeh and Azad, 2005]. The necklace length may be fixed or variable for different edge necklaces. For fixed necklace length, the network called Regular Necklace-Hypercube (RNH) has the scalability $2^{n-1}(nk+2)$, where n is the number of dimension and k is the necklace size. A (n, k) RNH has a total number of nodes equal to $2^n + (n2^{n-1})$ x k with the maximum degree of 2n. Few of the processors have a constant degree equal to 2. It has a small diameter of n + k and a bisection width twice that of the standard hypercube i.e b= 2^n.

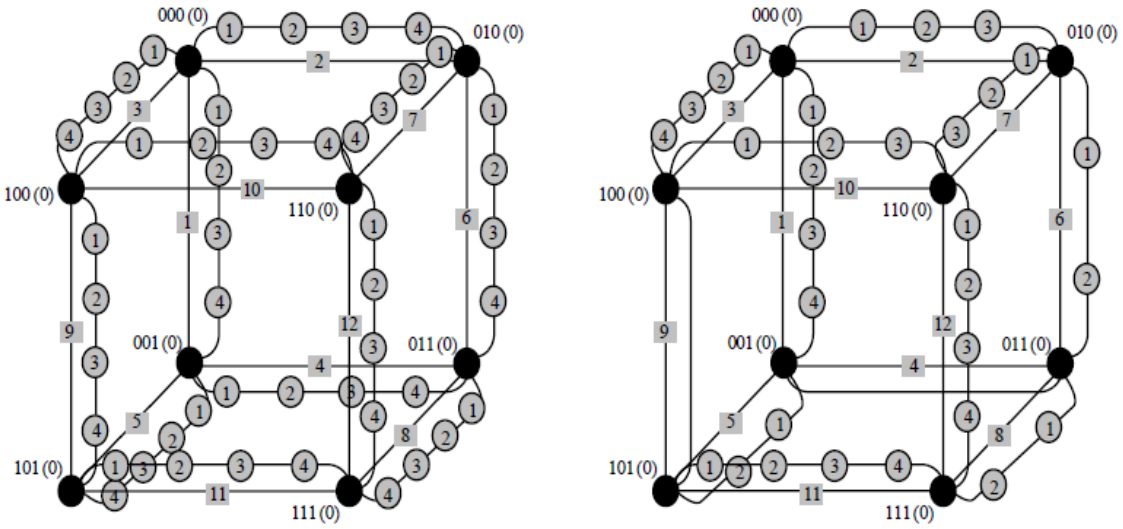


Figure. 2.7: Necklace Hypercube

2.6.12 DOUBLE-LOOP HYPERCUBE (DHL)

The DLH (m, n) where m is the number of bits in Johnson code and n indicates the dimension is proved to be a better scalable architecture. It combines the positive features of the hypercube topology such as small diameter, high connectivity, simple routing and constant node degree of a new double-loop topology. The DLH adopt the hybrid coding combining Johnson code and Gray code. With the hybrid coding, it is easy to implement the efficient routing algorithms. The adjacent nodes in the DLH differ exactly by one bit. The number of nodes in DLH (m, n) is 4 m x 2^n whereas the diameter is m+n+1 [Youyao et al, 2008]. Claims that the DLH network can maintain a constant node degree of n+3, independent to the increase in the network size. It has better scalability and the extensibility of size scaling is 4 x 2^n.

2.6.13 TWISTED HYPERCUBE

A twisted hypercube has the same structural complexities of the hypercube. It preserves the attractive properties of the hypercube and improves on the communication time by reducing the diameter by a factor of two. A twisted hypercube of dimension n denoted by TQn as an undirected graph consisting of 2^n vertices labeled from 0 to 2^n -1. The Total number of nodes

in twisted hypercube of dimension n is 2^n. The degree is n and diameter ⌈ (n+1)/2⌉ . Twisting of hypercube has an extra advantage of reducing the diameter without changing the degree of nodes. Though the routing in this case becomes a little more complicated the existing a suitable algorithm for routing in a twisted hypercube [Yang, Dong, Yang and Cao, 2011]. Also, the number of link pairs exchanged to reduce the diameter by a given value is much smaller compared to that given in Figure 2.8.

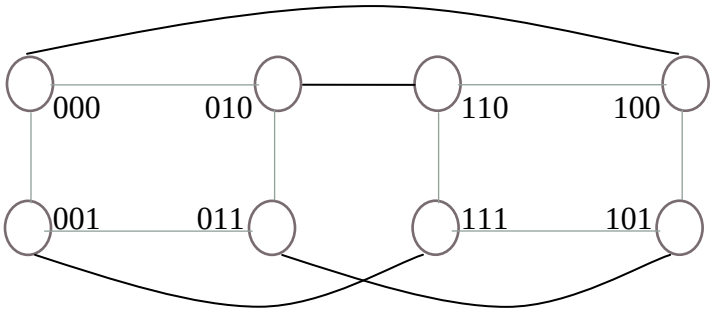


Figure. 2.8: Twisted Hypercube

2.6.14 FOLDED CROSSED CUBE (FCC)

The FCC is a new cube based interconnection topology for parallel system with better features such as reduced diameter, cost and improved broadcast time, better fault tolerance and better message traffic density [Adhikari and Tripathy, 2010]. It is constructed by connecting each node to a node farthest from it. The FCC is a graph Fr (V, E), where V represents a set of vertices and E represent a set of links as shown in Figure 2.9. The node connectivity of FCC is (n+1) with a number of nodes 2^n. The diameter of the FCC is $\lceil n/2 \rceil$ and bisection width is $2^n/2$.

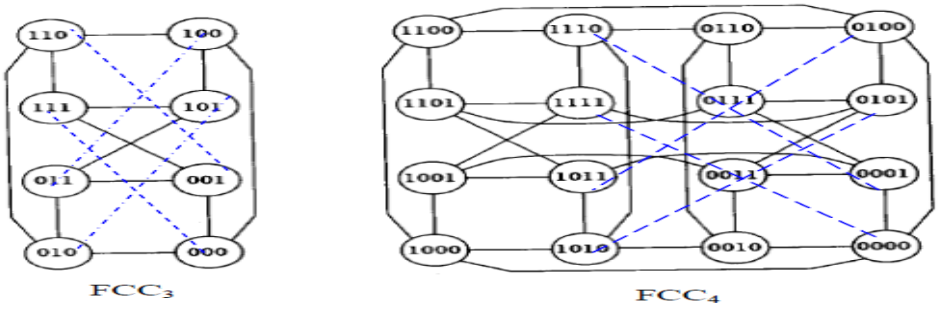


Figure. 2.9: Folded Crossed Cube

2.6.15 STAR CROSSED CUBE (SCQ)

The Star Crossed Cube (SCQ) is observed to be superior to other contemporary networks in terms of various topological parameters such as diameter, cost average distance and message traffic density [Adhikari and Tripathy, 2014]. The SCQ is the product of the Crossed Cube and Star Cube. In a star, each vertex is replaced with a CQ. The SCQ (3, 3) along with the parts, is shown in Figure 2.8. The node connectivity of SCQ is [m+n-1] with a number of nodes n! 2m. The diameter of the SCQ is [(m+1)/2 + ⌊3 (n-1)/2⌋].

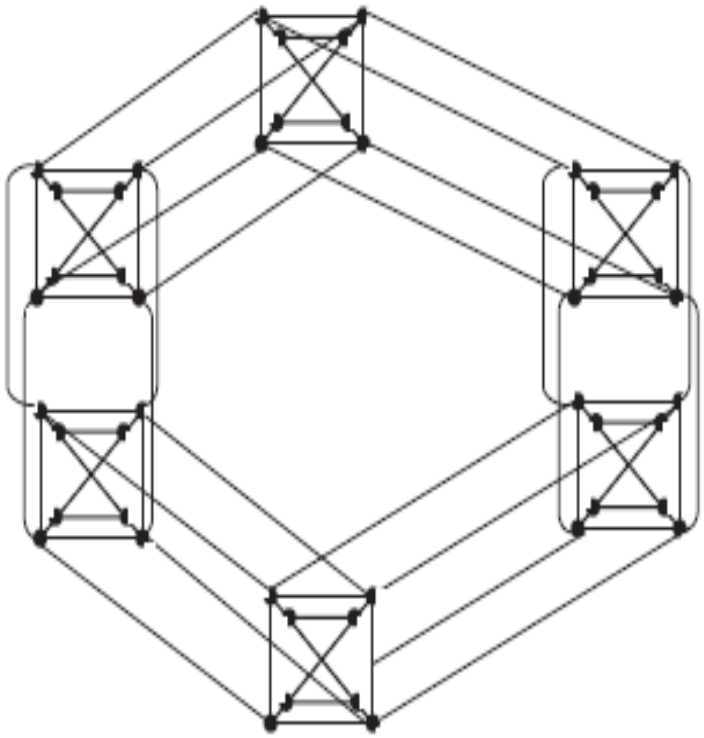


Figure. 2.10: Star Crossed Cube

Table 2.1: Summary of Cube based Interconnection Network Characteristics

Networks	Nodes	Degree (d)	Diameter (D)	Cost	Bisection width(B.W)	Extensibility
$CCC_{(n)}$	$n.2^n$	3	(5n/2)-1	3[(5n/2)-1]	2^{n-1}	Exponential
$HC_{(n)}$	2^n	n	n	n^2	2^{n-1}	Exponential
$FHC_{(n)}$	2^n	n+1	$\lceil n/2 \rceil$	$\lceil n/2 \rceil$ (n+1)	2^n	Exponential
$CC_{(n)}$	2^n	n	$\lceil n+1/2 \rceil$	n $\lceil (n+1)/2 \rceil$	2^{n-1}	Exponential
$RH_{(1,n)}$	$2^{2^{n+1}}$	n+1	n+2	n^2 +3n+2	2^{1+n}	Exponential
$HCN_{(n,n)}$	2^{2n}	n+1	n+ $\lfloor n+1/3 \rfloor$ + 1	(n+1)(n+ $\lfloor n+1/3 \rfloor$ +1)	$2^{2n}/2$	Exponential
$DC_{(n)}$	2^{2n+1}	(n+1)/2	n+1	(n+1)(n+1)/2	2^{n-2}	Exponential
$MC_{(1,n)}$	2^n	n+1	2^{1+n}	$(n+1)2^{1+n}$	$2^{2n}/2$	Exponential
$FDC_{(n)}$	2^{2n-1}	(n+3)/2	n-1	(n-1)(n+3)/2	$2^n/2$	Exponential
$FMC_{(1,n)}$	2^n	n+1	2n-1	(n+1) (2n-1)	$2^{2n}/2 + 2^{2n}$+n-2	Exponential
$FCC_{(n)}$	2^n	n+1	$\lceil n/2 \rceil$	(n+1) $\lceil n/2 \rceil$	$2^n/2$	Exponential
NH	$2^n+(n2^{n-1})k$	2n	n + k	2n* (n+k)	2^n	Exponential
DHL	$4mx2^n$	n+3	m+n+1	(n+3) (m+n+1)	--	Exponential
TH	2^n	n	$\lceil (n+1/2) \rceil$	n* $\lceil (n+1/2) \rceil$	-	Exponential
SCQ	n!2m	m+n-1	(m+1)/2++ $\lfloor 3_{(n-1)/2} \rfloor$	(m+1)/2++ $\lfloor 3(n-1)/2 \rfloor$ *m+n-1	-	Exponential

2.7 REVIEW OF LINEARLY EXTENSIBLE MULTIPROCESSOR INTERCONNECTION NETWORKS

2.7.1 LINEAR ARRAY (LA)

It is one dimensional network having the simplest topology with n-nodes having n-1 communication links. The internal nodes have degree 2 and the termination nodes have degree1. The diameter is n-1, which is long for large n and the bisection width is 1. It is asymmetric network. Linear array are the simplest connection topology. As the diameter increases linearly with respect to n, it should not be used for large n. For every small n, it is rather economical to implement a linear array.

2.7.2 BINARY TREE (BT)

A binary tree is either empty or consists of node called the root together with two binary trees called left subtree and the right subtree. When h is equal to height of a binary tree then maximum leaves are equal to 2^h and maximum nodes are 2^{h+1}-1. In a binary tree network there is only one path between any two nodes. The binary tree is scalable architecture with a constant node degree and constant bisection width. In general, an n-level, complexity balanced binary tree should have $N=2^{n-1}$ nodes [Khan, Siddiqui and Samad, 2013]. The maximum node degree is 3 and the diameter is 2(n-1). But has a poor bisection width of 1. A binary tree of 15 nodes in four levels is shown in Figure 2.9.

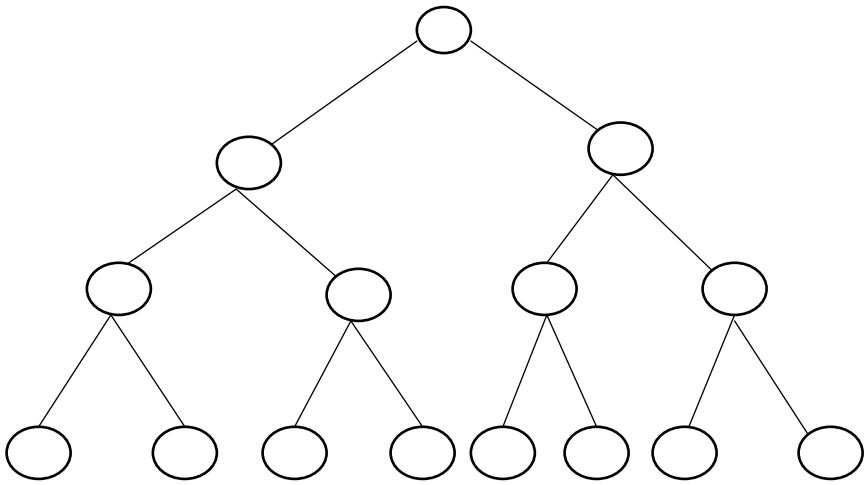


Figure 2.9: Binary Tree

2.7.3 RING (R)

This is a simple linear array where the end nodes are connected. It is equivalent to mesh with wrap around connections. The data transfer in a ring is normal one direction. A ring is obtained by connecting the two terminal nodes of a linear array with one extra link. A ring network can be uni-or bidirectional and it is symmetric with a constant. It has a constant node degree of d=2, the diameter is $\lfloor N/2 \rfloor$ for a bidirectional ring and N for unidirectional ring. A ring network has a constant width 2 [Yang et al., 2007]. The different performance parameters of Linearly Extensible Architectures are summarized in Table2.

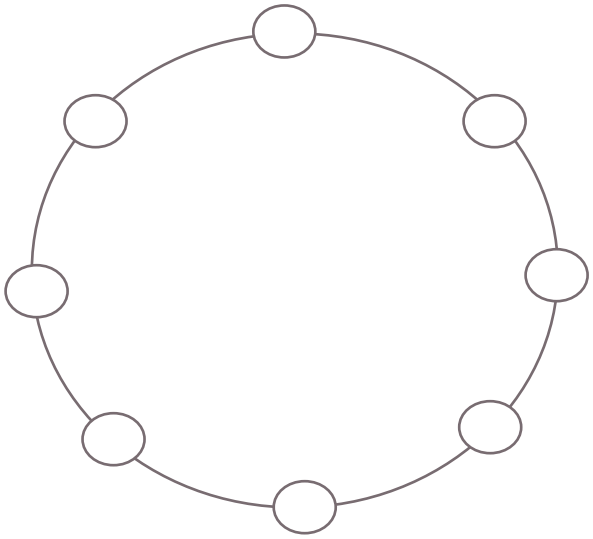


Figure 2.10: Ring

2.7.4 LINEARLY EXTENSIBLE TREE (LET)

A binary type network topology has been reported shown in Figure 4. The Linearly Extensible Tree (LET) architecture exhibits better connectivity, lesser number of nodes over cube based networks. The LET network has low diameter, hence reduce the average path-length traveled by all message and contains a constant degree per node. The LET network grows linearly in a binary tree like shape. In a binary tree the number of nodes at level n is 2n whereas in LET network the number is (n+1) [Samad, Rafiq and Farooq, 2010].

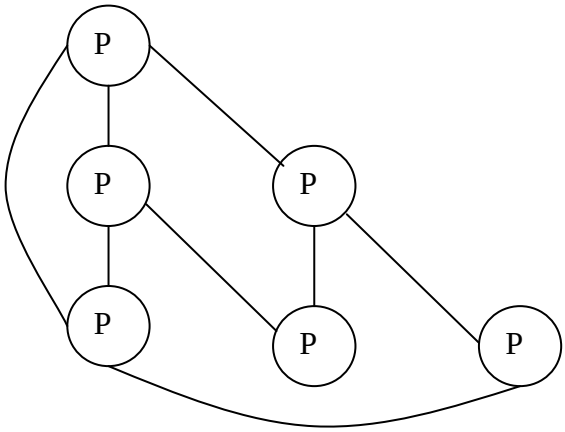


Figure 2.11: Linearly Extensible Tree (LET)

2.7.5 LINEARLY EXTENSIBLE CUBE (LEC)

The LEC network grows linearly and possesses some of the desirable topological properties topological properties such as small diameter, high connecting constant node degree with high scalability. It has constant expansion of only two processors at each level of the extension while preserving all the desirable topological properties. The LEC network can maintain a constant node degree regardless of the increase in size (Number of nodes) in a network [Samad, Rafiq and Farooq, 2010].

The number of nodes in LEC network is 2 * n for n > 0 whereas the number of nodes in the hypercube is 2n. The diameter of network is $\lfloor N \rfloor$. It has a constant node degree 4.

The LEC has a bisection width equal to N, as shown in Figure 2.12.

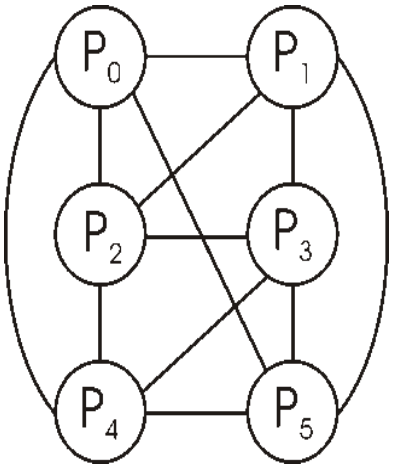


Figure. 2.12: Linearly Extensible Cube

Table 2.2: Summary of Linearly Extensible Interconnection Network Characteristics

Networks	Nodes	Degree (d)	Diameter (D)	Cost	Bisection width(B.W)	Extensibility
Binary Tree	2^{n-1}	3	2(n-1)	6 (n-1)	1	Linear
Linear Array	n	2	n-1	2(n-1)	1	Linear
Ring	n	2	$\lfloor n/2 \rfloor$	$2(\lfloor n/2 \rfloor)$	2	Linear
LET	2*n	4	$\sqrt{n}$	$4\sqrt{n}$	$2\log_2(n+2)$	Linear
LEC	2n	4	$\lfloor n \rfloor$	$4 \lfloor n \rfloor$	n	Linear

2.8 CONCLUSION

Form the above discussion and the comparative study made in Table 2.1, it may be concluded that:

- The hypercube node degree increases with $\log_n N$ and is also not appealing when N becomes large. Similarly, in Folded Hypercube there are 2^n nodes of degree n+1, which is not desirable.
- Networks diameters vary over a wide range. Although, the hypercube has small diameters but it is lacking in terms of scalability.
- The Folded Hypercube and Dual Cube have a good scalability, but still there is exponential extensibility, which is un-desirable.
- The Linearly Extensible Cube node degree is constant with linear extensibility because there are 2 n nodes.
- The LEC network has small diameter and it's more scalable from the other linear networks.

The core of all efforts to exploit the potential power of parallel systems is not only limited to design the efficient architectures, rather the second important issue is the balancing of the computational load over these networks. Load balancing is must to maximize processor

utilization, avoid any waste of resources, and to increase the overall performance of the system. Therefore, a number of scheduling schemes to balance the load have been discussed in chapter 3.

REVIEW OF SCHEDULING ALGORITHMS

3.1 INTRODUCTION

Strides in hardware as well as in software technologies have resulted in improved curiosity about the usage of large-scale parallel and distributed systems for database, real-time applications and also large-scale industrial applications. The main motivation to design massively parallel system is to run applications to enjoy continued speed increase in future hardware generations. Researchers have achieved speedups of more than 100x for some applications. However, this is typically achieved only after extensive optimization and tuning of algorithms so that more than 99.9% of the application execution time is in parallel execution. Therefore, it is essential that an application has the vast majority of its execution in the parallel portion for a massively parallel processor to effectively speed up its execution. The operating system and management of the concurrent procedures make up essential aspects of the parallel and distributed environments. One of the greatest problems in this kind of systems is the progression of useful methods for the distribution of the processes of a parallel program on numerous processors [Dhakal et al., 2003], [Chen et al., 2008], [Altman et al., 2011]. The important issue is how to schedule the procedures among transactions elements to attain certain performance aim; for instance reducing execution time, reducing communication delays, and increasing resource utilization. From a system's point of view this distribution option turns into an asset management issue and also works extremely well at the time of the design stages of multiprocessor systems. Process scheduling techniques are usually categorized into a number of subcategories as shown in Figure 3.1[Sharma and

Gahlawat, 2013], [Bansal et al., 2011], [Kaur and Kaur, 2013]. It should to be mentioned that all through this manuscript the terms job, process and task are widely-used interchangeably. Local scheduling carried out by the operating system of a processor that includes the assignment of processes to the time-slices of the processor [Samad et al., 2012]. Global scheduling, on the other hand, is the strategy of determining where to implement a process in a multiprocessor system. Global scheduling may be performed by a sole central authority or it may be distributed among the processing elements [Zaki et al., 1997], [Sharma et al., 2008], [Bertogna et al., 2009].

Many parts of the implementation of a parallel language affect application performance including communication among processors, the granularity of tasks, and the scheduling policy. A scheduling policy determines the schedule, the order in which parallel tasks are evaluated and assignment of parallel tasks to processors or processor cores. This assignment may be determined as a part of the compilation process by the runtime system or by a combination of the two [Andrade et al., 2008] [Barbosa and Moreira, 2011]. Parallel programming encourages programmers to express a large number of parallel tasks to run on different processors or processor cores. This ensures that parallel programs are portable i.e. programmers can develop programs independently of the number of processors and be confident that there will be enough parallel tasks to take advantage of whatever parallel hardware is available [Roy et al., 2012], [Waldspurger et al., 1995]. The length of any given task may be difficult to predict. Moreover, dividing the program into many small tasks should ensure that load can be evenly balanced among these processors. Small changes in the program input or compiler can also affect performance in different ways that are difficult to predict [Dhodhi et al., 2002], [Ghafoor and Yang, 1993]. Generally, it is not feasible to test programs on all possible inputs and hardware configurations. Therefore, one possible approach to understand the performance is to give bounds on the time or memory required by an application. For example, no scheduling policy can run faster than the optimal schedule with an unbounded number of processors. Our focus is to go on global scheduling techniques that are categorized into a couple of significant organizations: static scheduling and dynamic scheduling which is frequently known as dynamic load balancing [Zhang et al., 2009], [Dobber et al., 2009].

3.2 STATIC SCHEDULING

In static scheduling, the assignment of tasks to processors is carried out before program execution begins. Information regarding task execution times and processing resources is assumed to be known at compile time. A task is always executed on the processor to which it is assigned; that is, static scheduling methods are processor non-preemptive. Typically, the goal of static scheduling methods is to minimize the overall execution time of a concurrent program while minimizing the communication delays [Colajanni and Dias, 1998], [Kwok and Ahmad, 1999]. With this goal in mind, static scheduling methods attempt:

- Predict the program execution behavior at compile time (that is estimate the process or task, execution times, and communication delays).
- Perform a partitioning of smaller tasks into coarser-grain processes in an attempt to reduce the communication costs.
- Allocate processes to processors.

The major advantage of static scheduling methods is that all the overhead of the scheduling process is incurred at compile time and resulting in a more efficient execution time environment compared to dynamic scheduling methods. Static scheduling methods can be classified into optimal and suboptimal as shown in Figure 3.1 [Balter et al., 1998], [Bahnasawy et al., 2011]. Perhaps one of the most critical shortcomings of static scheduling is that, in general, generating optimal schedules is an NP-complete problem. It is only possible to generate optimal solutions in restricted cases (for example when the execution time of all of the tasks is the same and only two processors are used). NP-completeness of optimal static scheduling with or without communication cost considerations has been proven in the literature [Amalarethinam and Josphin, 2015], [Stigge et al., 2011], [Kafil and Ahmad, 1998]. Consider a simple example to demonstrate the difficulties in attaining general optimal schedules. Suppose that we have n processes with different execution times which are to be scheduled on two processing elements such as PE1 and PE2. Since the goal of a scheduling method is to minimize the completion time of a set of processes the scheduler must decide which process should be assigned to (or scheduled on) which Processing Element (PE) so that the overall completion time is minimized. In this case, the optimum schedule will ensure that the processing loads assigned to PE1 and PE2 are equal (with no

unnecessary delay periods). However, this problem is NP-complete since it can be easily mapped to the well-known NP-complete "set-partitioning" problem.

Heuristic methods as the name indicates rely on rules-of-thumb to guide the scheduling process in the right direction to reach a "near" optimal solution. For example the length of a critical path for a task is to be defined amongst several possible longest paths. No concurrent program can complete its execution in a time period less than the length of its critical path [Braun et al., 2001], [Fang et al., 2010]. A heuristic scheduling method may take advantage of this fact by giving a higher priority in the scheduling of tasks with longer critical path lengths. The idea is that by scheduling the tasks on the critical path first we have an opportunity to schedule other tasks around them thus avoiding the lengthening of the critical path.

It should be noted that there is no universally accepted figure or standard for defining a "good" solution or a degree of "nearness" to an optimal solution. The researchers often use a loose lower-bound on the execution time of a concurrent program (for example, the length of a critical path) and show that their method can always achieve schedules with execution times within a factor of this lower-bound [Roy et.al, 2012], [Yang et al., 2013].

In addition to the NP-completeness of optimal general scheduling algorithms, static scheduling suffers from a wide range of problems most notable of which are the following:

- The insufficiency of efficient and accurate methods for estimating task execution times and communication delays can cause unpredictable performance degradations [Lee 1991, Wang 1991]. The compile time estimation of the execution time of a program's tasks (or functions) is often difficult to find due to conditional and loop constructs whose condition values or iteration counts are unknown before execution. Estimating communication delays at compile time is not practical because of the run-time network contention delays.
- Existing task/function scheduling methods often ignore the data distribution issue. This omission causes performance degradations due to run-time communication delays for accessing data at remote sites.
- Finally, static scheduling schemes should be augmented with a tool to provide a performance profile of the predicted execution of the scheduled program on a given

architecture. The user can then utilize this tool to improve the schedule or to experiment with a different architecture.

3.3 DYNAMIC SCHEDULING

Dynamic scheduling is based on the redistribution of processes among the processors during execution time. This redistribution is performed by transferring tasks from the heavily loaded processors to the lightly loaded processors (called load balancing) with the aim of improving the performance of the application [Amalarethinam and Mary, 2011]. A typical load balancing algorithm is defined by three inherent policies:

- Information policy which specifies the amount of load information made available to job placement decision-makers.
- Transfer policy which determines the conditions under which a job should be transferred. The current load of the host and the size of the job under consideration (the transfer policy may or may not include task migration that is suspending an executing task and transferring it to another processor to resume its execution) is to be determine.
- Placement policy which identifies the processing element to which a job should be transferred.

The load balancing operations may be centralized in a single processor or distributed among all the processing elements that participate in the load balancing process. Many combined policies may also exist. For example, the information policy may be centralized but the transfer and placement policies may be distributed. In that case, all processors send their load information to a central processor and receive system load information from that processor. However, the decisions regarding when and where a job should be transferred are made locally by each processor. If a distributed information policy is employed, each processing element keeps its own local image of the system load [Abdelkader and Omara, 2012], [Zheng et al., 2010]. This cooperative policy is often achieved by a gradient distribution of load information among the processing elements. Each processor passes its current load information to its neighbors at preset time intervals resulting in the dispersement of load information among all the processing elements in a short period of time [Grosu and Chronopoulos, 2004], [Sahoo et al., 2013]. A distributed-information policy can also be non-

cooperative. Random scheduling is an example of non-cooperative scheduling in which a heavily loaded processor randomly chooses another processor to transfer a job. Random load balancing works rather well when the loads of all the processors are relatively high i.e. it does not make much difference when a job is executed [Cosenza et al., 2011], [Barbosa and Moreira, 2011].

The advantage of dynamic load balancing over static scheduling is that the system need not be aware of the run-time behavior of the applications before execution. The flexibility inherent in dynamic load balancing allows for adaptation to the unforeseen application requirements at run-time [Spies et al, 1996], [Li and Kameda, 1998], [Ali et al., 2000]. Dynamic load balancing is particularly useful in a system consisting of a network of workstations in which the primary performance goal is to maximize utilization of the processing power instead of minimizing execution time of the applications.

The major disadvantage of dynamic load balancing schemes is the run-time overhead due to:

- The load information transfer among processors.
- The decision-making process for the selection of processes and processors for job transfers.
- The communication delays due to task relocation itself.

3.4 TAXONOMY OF LOAD BALANCING

A hierarchical taxonomy for load balancing in general purpose parallel and distributed computing systems. As demonstrated in the Figure 3.1., scheduling algorithms are classified as Local vs. Global Scheduling. Local scheduling is performed by the operating system of a processor and consists of the assignment of processes to the time slices of the processor. On the other hand, global scheduling is concerned with the assignment of processes in a multi-processor system [Majumdar et al., 1992], [Daouas et al., 1995]. Because grid network consists of multiple processors that are geographically distributed, grid scheduling falls under global scheduling. Therefore, the present taxonomy is equally applicable for grid computing environment.

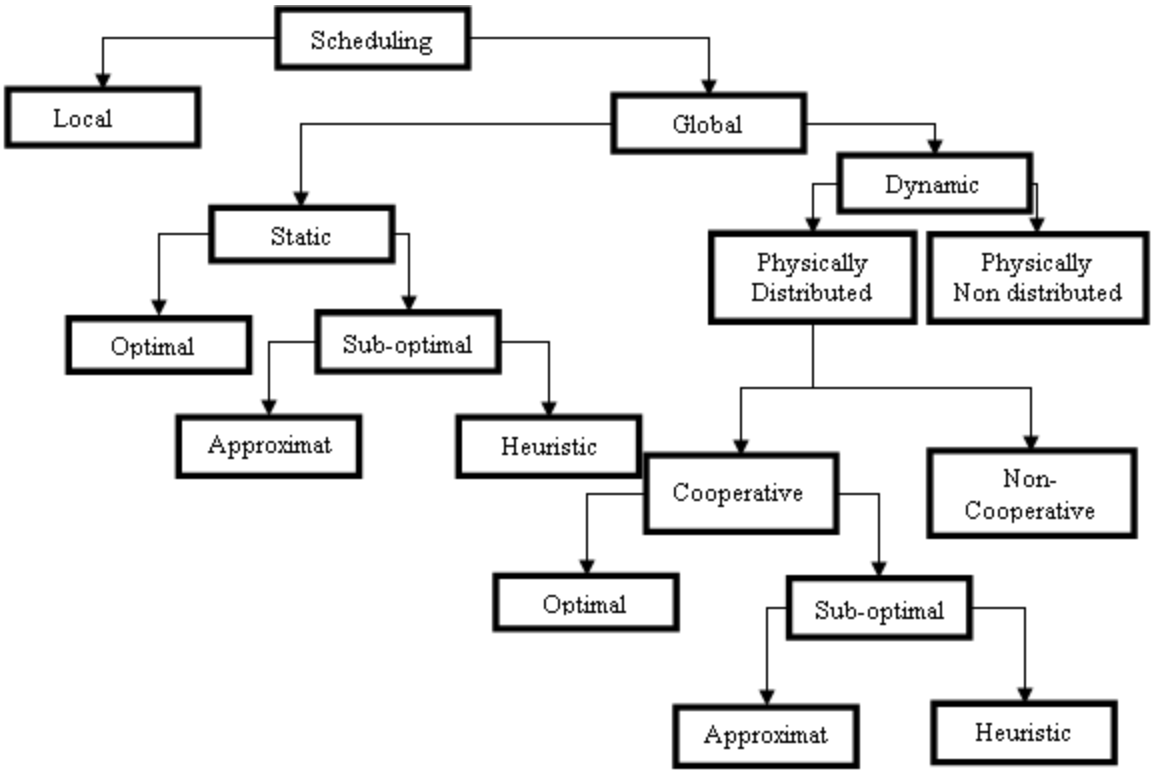


Figure 3.1: Classification of Load Balancing Scheduling Algorithms

3.4.1 TASK MIGRATION

Terminology:

- Task placement: non-preemptive transfer of process that has never run
- Task migration: preemptive transfer of process that has been executed
- Home node: site where the process originates
- Foreign process: process executing on a node other than its home node
- Freezing: when a process is migrated, its execution is interrupted (frozen) for the time it takes to move the process to another site.

Migration is useful for several reasons:

- load distribution, to equalize workload due to
  - transient periods of high load at some network nodes (transfer processes from temporarily overloaded node to another node)
  - In a system where process origination is always unbalanced (a few workstations consistently generate many processes)

- To return a workstation to its owner (migrate a foreign process back to its home node when owner returns)
- To improve communication (processes that communicate frequently can be moved to the same node; processes that access resources at a remote site may be moved to that site)

Issues in Task Migration:

For most scheduling algorithms task migrations carried out when the scheduler is invoked. Consequently, it is noticed that similar trends are obtained as with scheduler invocations [Karatza and Hilzer, 2002] [Attiya and Hamam, 2006]. The related issues in task migration could be listed as:

- State Transfer: Issues to consider here include
    - Cost: When a process is moved to another node it is frozen until the transfer is complete. There is the cost of a complete context switch, transfer, and other effect on the processes taken into consideration. (Processes that interact with the migrating process may timeout during the interval when it is frozen, and thus could be aborted)
    - Residual dependencies: These are resources that the original host node must maintain on behalf of the migrated process. For example, a process may transfer pages in virtual memory only as they are referenced. The original host may have to forward messages directed to the process at its original site.
- Location Transparency: Users aren't required to know where a task is executed. There should be no affect on the process such as process names, file names, etc. and must be independent of the host machine. A uniform name space is important to allow the process to maintain access to system resources, to continue to be able to communicate with other processes.
- Structure of the Migration Mechanism: It is the policy that decides when, where, and why transfer is to be made. However, policies can be changed without affecting mechanisms.

3.5 DESIGN ISSUES OF SCHEDULING ALGORITHMS

The scheduler is responsible for the following tasks. These tasks are important for efficient execution of applications on resources.

3.5.1 RESOURCE DISCOVERY

Resource discovery in distributed environment becomes a complex task as the resources are geographically distributed, heterogeneous in nature and maintained by different individual organizations. In general, resource discovery mechanisms use a database (for centralized approach) or a set of databases (for distributed approach) residing at different places.

3.5.2 RESOURCE SELECTION

Once the resources are discovered the next step of the scheduler is to select the number of resources required to execute the application with respect to meeting the user requirements such as deadline and budget or both.

3.5.3 TASK MAPPING

The next step of resource selection is to map the task to appropriate resources from the resource list to start their execution.

3.5.4 TASK MONITORING

The task may face unpredictable situations while executing the task by the resource such as resource failure, unavailability of input data and so on. In traditional parallel and distributed systems, the computational resources are usually managed by a single control point. The scheduler not only has full information about all running/pending tasks and resource utilization but also manages the task queue and resource pool [Sahoo et al., 2005], [Wang et al., 1997]. Thus it can easily predict the behaviors of resources and also to assign tasks to resources according to certain performance requirements. In distributed environments the scheduler does not have full control of the resources which makes it hard for the scheduler to estimate the exact cost of executing task on different sites. The computing resources are selected with respect to the application requirements and the status of the resources dynamically. These challenges depict unique characteristics of distributed environments and put significant obstacles to design and implement efficient and effective scheduling systems.

It is believed, however, that research achievements on traditional scheduling problems can still provide stepping-stones when a new generation of scheduling systems is being constructed [Simon et al., 2009], [Nguyen and Desideri, 2012].

3.6 TYPES OF DYNAMIC SCHEDULING

In a multiprocessor system the effect of a scheduling operation is observed on all the co-operating processors. There are different models of DS which decides the effect and co-ordination across the processors [Dobber et al., 2004], [Banicescu and Velusamy, 2002], [Attiya, 2004], [Beaumont et al., 2008], [Chandra and Shenoy, 2008]. The different models are fallow:

3.6.1 CENTRALIZED

Centralized load balancing policies are characterized by the use of a dedicated processor which also takes part in computation. This processor is also known as master processor or central scheduler responsible to make the entire load balancing decisions [Lin and Raghavendra, 1992]. The main number of nodes collects the global information about the state of the system and allots tasks to individual node. In this way it can improve the resource utilization by applying sophisticated algorithms. However, for large system consisting of 100 or 1000 of nodes the master processor becomes a bottleneck. Moreover, if the central processor fails the whole system stops working.

3.6.2 FULLY DISTRIBUTED

It is an alternative to centralized approach in which the load balancing decisions are carried out by all the processors of the system. Each node executes a scheduling algorithm by exchanging information with other nodes. It is therefore very costly for each node to obtain and maintained the dynamic state information of the whole system [Shivaratri et al., 1992].
Many variants of fully distributed schemes appear in the literature. However, there are number of drawbacks associated with them [Ishfaq and Ghafoor, 1991]. The first is that for large system (more than 100 processors) the optimal scheduling choice challenging to make even if the correct decisions are made it results a high control overhead at heavy load conditions. The second drawback is that the fully distributed algorithms use partial information about the state of the system for suboptimal decisions [Subrata et al., 2008].

A reduced amount of information results in a smaller range of scheduling options. Other problem with fully distributed schemes is of communication delays which may turns a correct scheduling decision into a wrong choice. Therefore, it may be concluded that fully distributed algorithm is a better option for small to moderate systems.

3.6.3 PARTIALLY DISTRIBUTED

Partially distributed or sometimes called as semi-distributed algorithms are proposed as a trade-off between centralized and fully distributed scheduling schemes [Ishfaq and Ghafoor, 1991]. The main idea is that the system is divided into different regions and thus the load balancing problem is divided into subtasks. Each region is generally managed by a single master processor using a centralized scheme. Master processors of each region may exchange information for balancing the load dynamically in the system.

3.6.4 SYNCHRONOUS VERSUS ASYNCHRONOUS

The fully distributed and partially distributed schemes may further be categorized as synchronous and asynchronous based on the instant at which load balancing operations are made. In synchronous schemes all processors involved in load balancing carry out balancing operations instantly. Each processor cannot proceed with normal computation until the load migrations demanded by the current operations have been completed. On the other hand, in asynchronous approach the running processor takes the load balancing decision independently. Each processor performs the balancing action regardless what the other processor doing at that same time. A number of synchronous and asynchronous load balancing algorithms have been discussed in the literature [Bahi et al., 2005].

3.7 REVIEW OF LOAD BALANCING SCHEDULING ALGORITHMS

Load balancing refers to getting the ideal utilization of processing elements to execute a number of parallel tasks so that to decrease the whole complete time frame in a multiprocessor system. These load-balancing algorithms can be extensively categorized based on different parameters that help to decide task migration such as centralized or decentralized, dynamic or static, periodic or non-periodic and individuals with thresholds or without thresholds. In a centralized load-balancing algorithm, the central scheduler will select the load balancing based on the global load information delivered from other processors [Vinay et al., 2011].

An efficient load valuation strategy offer adaptive decentralized load balancing technique in which every processor in the system retains track of the global load information and load balancing options may be produced by any kind of processor [Kremin and Kramer, 2005], [Azzoni, and Down, 2009], [Jain and Gupta, 2009]. Load evaluation and migration are some of the widely looked into suggestions in dynamic load balancing algorithms. Every load-balancing system involves issues such as the begin rule, information trade off guideline, task mapping and threshold policy because the efficiency of load balancing mechanism relies on sound task mapping [Mitchell et al., 2004].

There are varieties of methods to resolve non-uniform issues on multiprocessor systems depending on the several versions discussed above. The most appropriate strategies reported in the literature are discussed in the following section.

3.7.1 MINIMUM DISTANCE SCHEDULING (MDS)

This is another novel dynamic scheduling scheme for load balancing reported [Rafiq et al., 1999]. The algorithm operates on a minimum distance property which assures the minimization of the communication in distributing tasks among processors. In general, the performance of a multiprocessor system can be characterized by communication delay, distribution of load among the processors and scheduling overhead [Ravikanth et al., 1988], [Reddy, 1993]. Therefore, a close correspondence between the structures of the problem and the architecture of the processors is desired in order to minimize these overheads. When the problem graph topology is not known in priori, the mapping is done on the fly onto the processors. Thus, dynamic load balancing is essential for efficient utilization of highly parallel systems when solving non-uniform problems with unpredictable load estimates. The scheduling techniques may have certain constraints that may vary from application to application. The MDS scheme works to minimize the communication in distributing tasks among processors [LeMair and Reeves, 1993].

3.7.1.1 Minimum Distance Property

One of the important parameters for proper utilization of multiprocessor systems is the inter-processor communication costs which should be as small as possible. It necessitates some means to reduce these overheads. Therefore, to assign tasks on processors, a scheduling strategy must be designed which take care of minimization of execution and communication costs. Minimum distance is the property which assures the minimization of the communication in distributing subtasks and collecting partial results. Therefore, one of the method to sustain this property is to keep message path lengths to one hop. A scheduling scheme operates with this property minimizes overhead and ensures the maximum possible speedup. The property may be formally stated as: "If T and T_1 are the two tasks from a task tree of a given problem such that T is the parent of T_1 and if P and P_1 are the processors on which T and T_1 are scheduled, then, P should be directly connected to P_1 in the network." In the MDS algorithm, the adjacency matrix of the network is used to satisfy the minimum distance property [Ravikanth et al., 1988].

The general model of the dynamic load balancing is mainly based on the load balancing profitability determination at various sites in a multiprocessor Network [LeMair and Reeves, 1993]. When profitable, a scheduler is invoked which migrates tasks to achieve a more uniform distribution of load on processors. The donors (overloaded) and acceptors (underloaded) processors are identified based on a threshold value known as ideal load (IL). Each donor processor, during balancing, selects most suitable tasks (based on task dependencies) for migration thus maintaining minimum distance. Migration from donor processor is done to the directly connected acceptors. Thus, for every donor, there is a set of Minimum Distance Acceptors (MDA). Tasks are not allowed to migrate to acceptors which are outside this set. To perform the load balancing, the algorithm calculates ideal load (IL) value for each iteration which is used by load balancer as a threshold to detect load imbalances and make load migration decisions. The whole algorithm is demonstrated in Figure 3.2.

Algorithm MDS_sch ()

```
MDS ( )
{
/* Get the set of connected processors to the processor for which migration is being called i.e. number_P */
For (i =0; i < MAX_P; ++i)
{
If (connected (i, number_P, level))
Temp [k++] = i;
K--;
}
/ * Get the small loaded processor number */
Small = temp [0];
For (i = 0; i < MAX_P; ++i)
If (TG [temp[i] ] < TG [small])
Small = temp [j];
/* Transfer the load from p_number to the smallest loaded and connected processors */
While (TG [number_P]! = IL || TG [small]! = IL) {
TG [number_P] --;
TG [small] + =1 ;}}
/* Check the under loaded processors which are not connected. If any repeat the above procedure for the next level of connectivity */
}
/* Function used to find the maximum load on a processor */
Max (X [ ]){
Max = x [0];
For (i =0; i < MAX_P; ++ i)
If (x [i] > max) max = a [i];
Return (max);}
```

Figure 3.2: Minimum Distance Scheduling Algorithms

When implemented on Linearly Extensible Tree (LET) the MDS scheme shows that the network has good load balancing properties when considering problem structures having parallelism but non-uniform growth in various branches. The idea of managing domains

which decreases the over-head of the balancing method by the use of balancer, however, should not assure a balanced load for whole system that trade-off is illustrated in the scheduling strategies [Rafiq et al., 1995].

3.7.2 HIERARCHICAL BALANCING METHOD (HBM)

It is an asynchronous and decentralized approach of load balancing. It classifies the multi-computer system into a hierarchy of balancing domain. Each domain has a particular level of load balancing at different depth. Specific processors are designated to control the balancing operations at different levels of the hierarchy. The balancing process is invoked by the receipt of load update messages indicating an imbalance between lower level domains. Different imbalance thresholds can be set at different levels of the hierarchy. The HBM scheme distributes the load balancing responsibilities to all processors in the system. This scheme is effective to manage both the local load imbalance as well as excessive global imbalances [Bari and Meshram, 2013], [Manaullah et al., 2013].

Hierarchical multiprocessing scheduling algorithm makes use of a blend of space and time multiplexing to attain the preferred partitioning of CPU bandwidth within the scheduling tree. First, design to allot many CPUs to a tree node at the same time. To ensure that numerous threads by the processor sub tree should be implement in parallel. Second, the auxiliary proportional reveal thread scheduling scheme to carry out this CPU assignment to be able to attain preferred CPU service for the tree processors [Bhatti et al., 2011].

The Hierarchical scheduling includes two components i.e. an auxiliary scheduling and space scheduler. The auxiliary scheduler which is used to separator the recurring CPU bandwidth among the tree nodes proportional to their weights. The space scheduler is a scheduler that statically partitions the CPU bandwidth in integral variety of CPUs to allot to every processor in the hierarchy. Hierarchical scheduling algorithms have been created for uniprocessor and packet scheduling. Several of algorithms are not conveniently extensible to multiprocessor environment because they do not include the inherent parallelism of multiprocessor system when resource partitioning is taken into consideration, so they can lead to unbounded unfairness or hunger [Lipari and Bini, 2005], [Shin et al., 2008]. The issue of Parallelism are shown in Figure 3 .3.

Algorithm hier_sched ()

Root ⟶ Node

While node is not leaf do

Node ⟵ Gen_sched (Node) {Gen_sched that Selects a Node Child for Scheduling}

End while

Figure 3.3: The Hierarchal scheduling Scheme

3.7.3 GRADIENT MODEL (GM)

The gradient model is a regular load balancing technique in which each processor interacts with the instant acquaintances. The utilization of GM are involves moving backlogged tasks to close by best processors based on a stress gradient not directly proven by requirements from ideal processors. The algorithm is completely dispersed as well as asynchronous. Global balance is accomplished by consecutive refinements of numerous regular balances. The gradient model is developed in order to be unbiased of system topology. A best or even underutilized processor makes the load balancing actions by challenging a lot more work load. A need is content by the arrival of a task or tasks from additional greatly loaded processors [Lin and M. R. Keller, 1987]. The basic process is under loaded processors and notifies additional processors in the system of their state, consequently the overloaded processors react. This is an overloaded processor will send its extra load simply to one lightly loaded neighbor processor at the conclusion of one iteration of the load balancing algorithm [Bronevich and Meyer, 2008].

The scheme relies upon a couple of threshold parameters L1 and L2. A processor is known as overloaded while its load becomes more than L1, light if below the L2, and reasonable elsewhere. The Gradient Model scheme varies from the Dimension Exchange scheme in the sense. In GM, the load information of the whole main domain is known as in determining the location processor whereas in DE a single processor is recognized as at each iteration [Lin and Keller, 1987]. Depending on the gradient model, a dispersed load balancing algorithm for every node of a system can be created as demonstrated in Figure 3 .4.

Algorithm: GM_Schd ()

```
LOOP
Node i determines its internal loading state Pi
CASE state OF
Light: Setpi = 0. (Ignore the pressure information from neighbors)
Moderate: pi = 1 + min {pj} over all neighbor j
If P > wmax then pi Wmax (*saturation*)
Heavy: pi - 1 + min {p} over all neighbor j
If Pi> Wmax
then pi- Wmax (*saturation*)
else if min {pi} < pi,
then transfer one task to node j,
where pj is the minimum
END CASE
Broadcast pi to all neighbors, if pi has changed from last update
END LOOP
```

Figure 3.4: The Gradient Model Scheme

3.7.4 TWO ROUND SCHEDULING (TRS)

The basic approach in MDS is to optimize the load balancing among processors under the constraint of the need to keep message path lengths to one hop and thus satisfying the minimum distance property. Migration from donor processor is done to the directly connected acceptors. Thus, for every donor, there is a set of Minimum Distance Acceptors (MDA). Tasks are not allowed to migrate to acceptors which are outside this set. Therefore, a more dynamic nature of algorithm is required to make the networks fully balanced, which takes into consideration those processors also which are not directly connected [Samad et al., 2012].

A TRS scheme is for solving load balancing problem with unpredictable load estimates. The TRS algorithm works as an extension of MDS and named as Two Round Scheduling scheme. It is dynamic in the sense that no priory knowledge of the load is assumed. TRS

scheme takes into consideration those acceptor nodes which are not connected directly to donor node. There may be more than one path between the donor and acceptor processors which require multi-hop. However, large number of hopes gives minimum load imbalance and hence, improved value of LIF is obtained. Therefore, the proposed TRS algorithm has a constraint in the scheduling to consider only one processor as intermediate node between donor and acceptor nodes. To perform the load balancing, the algorithm calculates ideal load value for each iteration of the task structure in the same manner as calculated in MDS. The pseudo code for this algorithm is shown in Figure 3. 5.

Algorithm: TRS_Sch ()

```
TRS ()
{
/* Check the connectivity of processor I with J.  The connectivity level is 1 or 2 */
Int connected (int i, int j, int level) /* returns true if Processors i, j are connected */  {
 /* Printf ("\n node %d is connected to %d: %d", i, j, adj [i] [j]); */
 If (level = = 1)
 Return adj [i] [j];
 For (int k = 0; k < PMAX; k++)
{
 If (k = = i || k = = j) continue;
 If (connected (i, k, 1) && connected (k, j, 1))
 {
/* Printf ("\n node %d is connected to %d through %d", i, j, k); */
Return 1;
 }
 }
 Return 0 ;}
 End of procedure
```

Figure 3.5: The Two Round Scheduling Scheme

3.7.5 TREE WALKING ALGORITHM (TWA)

The Tree Walking Algorithm (TWA) is a parallel scheduling algorithm for tree organized interconnection network. This algorithm is a scalable scheduling algorithm that makes use of the global load information to maximize load balancing. Simultaneously, this algorithm reduces the variety of tasks to be shifted as well as the variety of connectivity. It balances the load perfectly and also efficiently decreases the processor idle time. [Bojańczyk and Colcombet, 2006].

There are five steps of TWA algorithm such as: first step is to be implemented only at the system setup time although not in every single system phase. Step 2 and 3 gather the system load information. In step 2, the amount of tasks is counted with a parallel decrease operation. Simultaneously, every node records the variety of tasks in the sub tree and its children's sub trees, if virtually any. In step 3, the root computes the average variety of tasks per node after which broadcasts the amount to each node to ensure that each one node understands in the event that the sub tree is over-loaded or under-loaded in step 4. If the quantity of tasks can never be uniformly split by the variety of nodes the outstanding tasks are uniformly split by the variety of nodes possess another task. In step 5, the work load is traded to ensure that at the conclusion of the system stage every node possesses the virtually identical variety of tasks [Bojańczyk and Colcombet, 2005]. Then, each one processor elapses independently and continues to the subsequent user phase but there is no global termination condition.

3.7.6 ADAPTIVE AND HIERARCHICAL TASK SCHEDULING ALGORITHM (AHS)

The adaptive and hierarchical task scheduling scheme (AHS) for multi-core clusters wherein work-stealing and work-sharing are adaptively accustomed to attain load balancing. The work-stealing and work-sharing are two fundamental paradigms for dynamic task scheduling algorithm [Wanga et al., 2014]. The Work-stealing continues to be popular in task-based parallel programing languages and models particularly on shared memory systems. Nevertheless, large inter-node connection expenses obstruct work-stealing from becoming instantly carried out on distributed memory systems. The AHS Scheme addresses this problem with the following methods for instance preliminary partitioning which decreases the inter-node task migrations and hierarchical scheduling scheme which carries out work-stealing inside a node before going across the node boundary and adopts. The work-sharing

to overlap computation as well as connection at the inter-node degree and also hierarchical and centralized manage for inter-node task migration which enhances the effectiveness of target choice and cancellation identification.

The AHS possesses used with current work-stealing schemes on a sixteen-node multi-core cluster that AHS outperforms current schemes. Wherein a global scheduler can make a primary partitioning of tasks with regards to the pattern of task parallelism and cooperates with local schedulers by important message passing. Work-stealing is executed by the local schedulers to balance load between worker threads on a cluster node and also work-sharing is employed along with work-stealing to attain load balancing between the cluster nodes [Tharani and K. Deepa, 2012]. The work-sharing offers overall performance advantage and AHS outperforms the present work-stealing schemes with true programs. In long term, we want to check AHS in a wide scale regard with additional cluster nodes and with various other technological extensive applications. These kinds of assessments will permit us to much better examine the conduct of AHS. A complete overview of AHS algorithm is depicted in Figure 3.6.

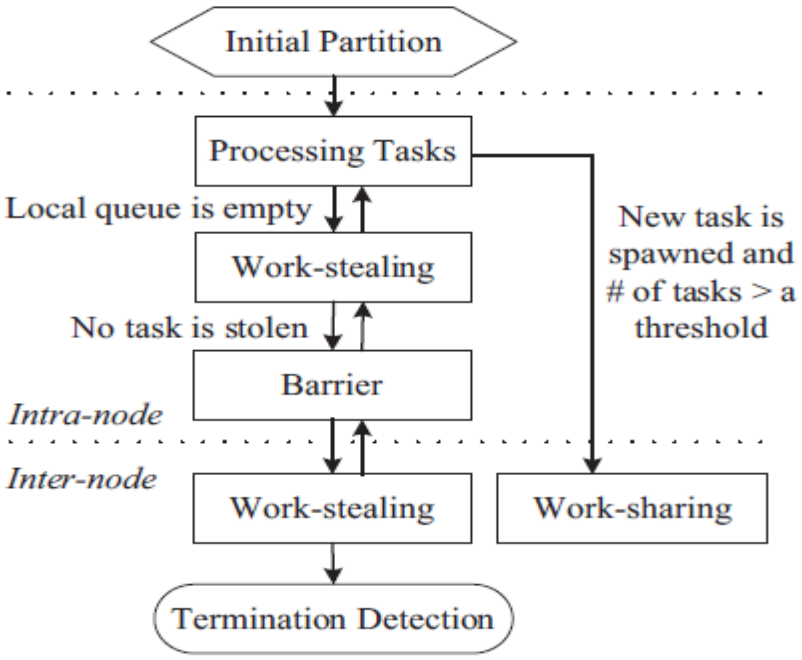


Figure 3.6. Overview of AHS scheduling algorithm

3.7.7 ITERATIVE GREEDY ALGORITHM (IG)

The iterated greedy algorithm starts from heuristically designed way out of the issue. An iteration-based tactic to achieve much better options from one era to the subsequent. The iterative greedy algorithm has three simple measures: destruction, construction and also acceptance criterion as shown in Figure 3.7. In destruction phase, certain components are randomly recovered from the incumbent way out of get a half solution. Later on, a total applicant remedy is reconstructed by utilizing a greedy development heuristic in the development phase. One a recently reconstructed solution continues to be received an acceptance criterion is working to choose whether or not it is going to exchange the incumbent solution or perhaps not [Page and Naughton, 2005]. These steps are performed respectively until a resting criterion is satisfied. The purpose of IG is to reduce the entire sum of performance, communication costs and impression is to improve the excellence of the assignment in an iterative manner using outcomes from earlier iterations [Zomaya et al., 2001]. The IG algorithm performance is examined with parallel to three finishing algorithms on several randomly produced mapping issues. Computational outcomes exhibited the priority of the IG algorithm suitable effectiveness. The benefit of IG algorithm it offers less parameters which need to be tuned than the contending algorithms which is somewhat simple, conveniently implementable algorithm when compared with HPSO algorithm that is the revolutionary way for the problem considered.

Algorithm: Iter_Gre_Alg ()

```
Procedure IG
{
L0 = Generate Initial Solution
L= Local Search (L0)                    % optional
    Repeat
Lp = Destruction (L)
Lc = Construction (Lp)
L *= Local Search (Lc)                  % optional
L = Acceptance Criteration (L, L *)
Until terminated Condition met
}
```

Figure 3.7: The common outline of IG

3.7.8 GENETIC ALGORITHM (GA)

A genetic algorithm (GA) is a search heuristic that mimics the process of natural selection. This heuristic (also sometimes called a metaheuristic) is routinely used to generate useful solutions to optimization and search problems. Genetic algorithms belong to the larger class of evolutionary algorithms (EA) which generate solutions to optimization problems using techniques inspired by natural evolution such as inheritance, mutation, selection and crossover. In a genetic algorithm, a population of candidate solutions (called individuals, creatures, or phenotypes) to an optimization problem is evolved toward better solutions. Each candidate solution has a set of properties (its chromosomes or genotype) which can be mutated and altered; traditionally, solutions are represented in binary as strings of 0s and 1s but other encodings are also possible.

The evolution usually starts from a population of randomly generated individuals and is an iterative process with the population in each iteration called a generation. In each generation, the fitness of every individual in the population is evaluated; the fitness is usually the value of the objective function in the optimization problem being solved. The more fit individuals are stochastically selected from the current population and each individual's genome is modified (recombined and possibly randomly mutated) to form a new generation. The new generation of candidate solutions is then used in the next iteration of the algorithm. Commonly, the algorithm terminates when either a maximum number of generations has been produced or a satisfactory fitness level has been reached for the population. A typical genetic algorithm requires:

- A genetic representation of the solution domain
- A fitness function to evaluate the solution domain

A standard representation of each candidate solution is as an array of bits. Arrays of other types and structures can be used in essentially the same way. The main property that makes these genetic representations convenient is that their parts are easily aligned due to their fixed size which facilitates simple crossover operations. Variable length representations may also be used but crossover implementation is more complex in this case. Tree-like representations are explored in genetic programming and graph-form representations are explored in evolutionary programming; a mix of both linear chromosomes and trees is explored in gene

expression programming. Once the genetic representation and the fitness function are defined, a GA proceeds to initialize a population of solutions and then to improve it through repetitive application of the mutation, crossover, inversion and selection operators.

In recent times, two hybrid genetic algorithms which designed as Critical Path Genetic Algorithm (CPGA) and Task Duplication Genetic Algorithm (TDGA). The CPGA is based on easy methods to utilize the best useful precious time of the processor economically and reschedule the critical path nodes to reduce their start time. Thereafter, several fitness functions are used one after the other. The first fitness function is related to reducing the entire execution time schedule length, and second a winch is related to pleasing the load balance. The TDGA is based on task duplication principle to reduce the communication overheads [Guzek et al., 2014].

3.7.9 DYNAMIC CRITICAL PATH DUPLICATION (DCPD) ALGORITHM

The Dynamic Critical Path Duplication (DCPD) scheduling algorithm is to schedule parallel tasks by taking into account the system heterogeneity the network bandwidth and the communication requirements of the applications. It determines the critical path of the task graph and selects the next node to be scheduled in a dynamic fashion. Therefore, DCPD scheduling algorithm could bring out the improved performance in terms of scheduling length and utilization. The algorithm initially needless to say each task is allotted to a single digital processor which the communication over-head between tasks is the average communication rate time.

3.7.10 MODIFIED CRITICAL PATH SCHEDULING (MCP) ALGORITHM

Modified Critical-Path (MCP) algorithm schedules Directed Acyclic Graphs (DAGs) with communication costs onto a bounded number of processors. A DAG consists of a set of nodes $\{n_1, n_2, ..., n_n\}$ connected by a set of edges, each of which is denoted by $e(n_i, n_j)$. Each node represents a task and the weight of a node $w_n(n_i)$ is the execution time of the task. Each edge represents a message transferred from one node to another node, and the weight of the edge $w_l(n_i, n_j)$ is equal to the transmission time of the message [Lemair and Reeves, 1993]. When two nodes are scheduled to a single processing element (PE) the weight of the edge connecting them becomes zero. A DAG also has two virtual nodes of zero weight: a start

node that is the start of the program and an end node that is the end of the program as shown in Figure 3.8 [Rajak et al., 2013], [Kwok and Ishfaq, 1996].

Algorithm: MCP ()

The Original MCP Algorithm

Step 1. Compute the ALAP time of each node.

Step 2. For each node, create a list which consists of the ALAP times of the node itself and all its descendants in an ascending order, sort these lists in an ascending lexicographical order, and create a node list according to this order.

Step 3. Schedule the first node in the list to the PE that allows the earliest execution, using the insertion approach. Remove the node from the list and repeat Step 3 until the node list is empty

Figure 3.8: Modified Critical Path Scheduling Algorithm

3.7.11 HYBRID DYNAMIC PARALLEL SCHEDULING ALGORITHM (HD-PSA)

The Hybrid Dynamic Parallel Scheduling Algorithm (HD-PSA) is specifically created to Chained Cubic Tree (CCT) for load balancing on interconnection networks. The effectiveness of the HD-PSA techniques and the beneficial options that come with the topologies are accepted through the analytical as well as trial and error assessment of many performance metrics, including execution time, load balancing accuracy, communication cost, number of tasks hops and tasks locality [Rosemarry et al., 2012]. When compared with the popular TWA on tree, HD-PSA attains possibly better or convergent performance. Load balancing is accomplished by both at the same level of accuracy within convergent execution time intervals [Mahafzah and Jaradat, 2010]. The communication cost is extremely influenced by the variety of processors where for lower and reasonable sizes.

3.7.11.1 Hybrid Dynamic Scheduling

The hybrid dynamic scheduling algorithms strategy used to balance the workload while taking into account the memory constraints. Let us first define the notations used in hybrid dynamic algorithms. For a node J just extracted from the local pool of ready tasks (by the master processor of J), we define:

- ncand: the number of candidates.
- {p1……,pncand}: the set of candidate processors initially sorted by increasing workload.
- Wmaster: the computational cost (number of floating-point operations) of the master task.
- Wslaves : the computational cost associated with the sum of all the slave tasks.

For each candidate processor pi, we know/compute:

- The quantities memi, bufi and facti
- Its workload loadi.

Figure 3.9 presents the main steps of our hybrid dynamic scheduling approach. Note that since violating a constraint related to bufi or memi would lead to a failure we never relax them during the algorithm. At Step 1, we compute an advised maximum number of slaves nlim used during a first attempt to balance the work over a subset of the nlim least loaded processors. nlim is designed to limit the number of slaves. In practice it is set by considering that the work given to each slave should be related to the master's work:

$$nlim = \min(ncand, \max(W_{slaves} / qW_{master}; 1)); \text{ with } q > 0. \eth \quad \ldots\ldots (1)$$

The larger is the parameter q, the smaller is the number of eligible slaves during Step 1 of Figure 3.9. Hence for large value of q we will authorize our scheduling to assign larger tasks to slaves. Step 2 of our algorithm is performed only when Step 1 did not succeed in distributing all work because of memory constraints. Processors are then added one by one during Step 2 with the objective of mapping the remaining work on up to ncand processors, while saturating the memory constraints. At Step 3, we then suppress the memory constraint relative to facti and redistribute the remaining work.

Algorithm: HDS ()

Receive information related to workload and memory.

1. Try to balance the workload on a maximum of nlim processors

2. If Step 1 did not succeed (the nlim processors are saturated in memory and work remains to be mapped) then add new slaves one by one.

3. If Steps 1 and 2 did not complete the mapping then suppress the memory constraint on the size of the factors, facti, and try to balance the remaining work onto the candidates.

Figure 3.9: The Hybrid Dynamic Scheduling

3.7.12 DEPTH-LIMITED SEARCH SCHEDULING ALGORITHM (DLS)

The depth-limited search (DLS) method is almost equal to depth-first search (DFS) but DLS solves the infinite state space problem because it bounds the depth of the search tree with a predetermined limit 1. Nodes at this depth limit are treated as if they had no successors. Thus, its time complexity is $O(b^l)$ and its space complexity is O(bl) (b is the branching factor). Note that DPS can be viewed as a special case of DLS with an infinite limit $l= \infty$ (in a sense, DLS is a more general approach). Also note that DLS is still incomplete (the given limit may be less than the depth of the solution) and nonoptimal (it may be greater). However, in order to have an informed upper bound of the optimal depth limit, there can reasonably use the diameter of the state space (i.e., the longest shortest path between any two nodes defined by the problem).

The DLS algorithm will not allocate focal points depending on the critical path (CP). It works comprehensive set matching of nodes to processors at each one phase to discover the most vital node. The concept of the DLS algorithm is to apply a composite parameter DL to choose a node with a greater static level as well as a lesser known begin time to schedule. Nevertheless, it should be mentioned that the level of the elected node is probably not the largest and also the start time might not be the original among all the ready nodes. It is the refined variance between the DLS algorithm as well as the ETF algorithm (observe that the ETF algorithm attempts to schedule a node which could begin previously and also breaks links by utilizing the static levels). At the start of the scheduling method, the DLs of ready nodes are dominated by the SLs since the ready nodes have been in greater levels in the task graph; and their begin times could be compact. However, while scheduling the nodes in smaller level (say, the exit nodes) the DLs of the ready nodes are dominated by their begin occasions on the processors because their SLs are compact while their begin times are big. This shows the problem in the conduct of the DLS algorithm. A node with a big SL may be scheduled first although its begin time is not compact. This will likely prevent the initial scheduling of more vital nodes.

Algorithm: DLS ()

```
DLS (node, goal, depth) {
  If (depth >= 0) {
    If (node == goal)
     x=goal if (goal=depth)
       Return node
    For each child in expand (node)
      DLS (child, goal, depth-1)
  }}
```

Figure 3.10: The Depth-Limited Search Algorithm

3.8 CONCLUSION

Motivated by the linear extensible properties of LEC and its performance analysis when TRS scheduling schemes is applied on it, a new cube like interconnection topology named Linear Crossed Cube (LCQ) has been proposed and analyzed. The proposed architecture with topological properties is discussed in the next Chapter 4. A new dynamic scheduling scheme has been devised and implemented to evaluate the performance of the proposed LCQ. The simulation results are discussed in Chapter 5.

From the above review, it is apparent that myriad of multiprocessor scheduling strategies exist which can be applied to specific structure of programs and specific system architectures. An optimal scheduling can be made based on some objective functions to enhance the performance of overall system. In general the following remarks can be highlighted.

- Static approaches are easier to implement and have minimal runtime overhead. However, these schemes are not applicable for parallel systems where computing resources and communication network/traffic are not known in advance.
- Dynamic approaches result better performance but at the cost of high overhead.
- De-centralized schemes are costlier than centralized because it is very difficult to obtain and maintained the dynamic state information of the whole system by individual nodes.

- TRS scheme is better than MDS scheme in terms of load distribution. However, it doesn't take care of balancing time and heavily depend upon the one-to-one connection of each node.
- The HBM strategies depend heavily on the system interconnection topologies. It is well structured for tree type architectures.
- The GAs are extensively counted as beneficial meta-heuristics for receiving exceptional solutions for wide selection of combinatorial optimization issues including the task scheduling issue. It's reducing the entire execution time schedule length and pleasing load balancing.

LINEAR CROSSED CUBE NETWORK

4.1 INTRODUCTION

The demand for higher computation speed and the indications of saturation in built-in circuit technology has given a flip to the development of multiprocessor systems. The logic for using multiprocessor is to develop efficient interconnection network by simply linking multiple processing elements known as nodes. The multiprocessor technique to parallelism is an extremely generic and also versatile one, nevertheless to great extent its triumph depends on the interconnection topology. In this approach, multiple nodes are widely-used to work concurrently for a given program so that the total execution time is reduced. Multiple processors in such a system are arranged to form an interconnection network. There has been a lot of research for designing the appropriate topology of interconnection networks for massively parallel computer systems [Parhami et al., 2000], [Zhang et al., 2002], [Shi and Srimani, 2005]. However, these topologies have been designed to achieve specific goal and there is no consensus on the best network organization [Kim and Veidenbaum, 1999]. Some variations of topologies focus on the reduction of the diameter [Khan et al., 2013], [Samad et al., 2010], a variety of them highlighted the design of easy routing as well as communication algorithm [Li et al., 2002]. Scalability has always been a significant problem to examine the overall performance of interconnection networks. Nevertheless, it can't be evidently stated that which interconnection network is carrying out desirable performance by contemplating almost all the topological parameters. Many large-scale multiprocessor systems have been

developed with their own topologies [Kwak and Jhon, 2007]. However, the essential purpose is that a specific topology requires exceptional characteristics for instance lesser degree, smaller diameter and high fault tolerance. So, the choice of the topology of the interconnection network is vital in the design of a conventional parallel system which may affect the overall system performance.

Analysis of parallel computer interconnection topology possesses noted large utilization of cube based topologies. A plethora of cube based multiprocessor networks have been cited which exhibit the excellent characteristics of this kind of multiprocessor systems [Esfahanian and Sagan, 1993], [Ghose and Desai, 1995], [Zhang et al., 2002]. Many interconnection networks such as trees and multi-dimensional meshes can be embedded in the cube. In general, there are two major categories of interconnection networks: cube based architectures and linear architectures. The first is cube based architectures that have a rich interconnection topology. The well-known interconnection network is binary hypercube or n-cube that has been widely utilized in the design of parallel systems. Numerous different versions of hypercube architecture are revealed in the literature. Some examples are: Folded Hypercube (FHC) [Amway and Latifi, 1991], Metacube (MC) [Peng and Chu, 2002], Folded Metacube (FMC) [Adhikari and Tripathy, 2009] as well as Folded Dualcube (FDC) [Adhikari and Tripathy, 2008] and so on. The popular downsides in such kind of networks are the increasing number of communication links for every node along with the rise in the whole number of nodes in the system which eventually enrich the complexity of these interconnection networks [Liu et al., 2004], [Mohanty et al., 2008]. Consequently, effort is made to accomplish the performance evaluation of numerous interconnection networks by considering their topological properties. The second class of parallel interconnection network is the linearly extensible networks which are simple architectures such as Ring, Linear Array, and Linearly Extensible Tree etc. [Rajput and Kumari, 2012]. The complexity of linear type networks is comparatively lesser as compared to cube based networks because they do not have exponential expansion. An additional class of multiprocessor architecture is hybrid architecture which anticipates the salient features of cube based architecture as well as they are less complex and conveniently extensible. Linearly Extensible Cube (LEC) is an example of such architecture [Samad et al., 2010]. The LEC architecture is inlayed the acceptable

characteristics of hypercube architecture and has an abundant connectivity and also maintains symmetry when prolonged into greater level. The extensibility of LEC is linear still it has smaller bisection width when extended to larger level of architecture and could not be considered a good candidate with high fault tolerance. Some other parameters to examine the overall performance of these systems are regularity, number of processors, degree and diameter. The numerous topologies have already been recommended and enforced to enhance the overall performance of systems. However, not a single topology states to possess all the suitable topological capabilities. For instance the Crossed Cube architecture indicated by CC is resulting from the hypercube to possess a compact diameter that is virtually half of its parent architecture. Nevertheless, the CC creates no improvement in the hardware cost as compared to the hypercube [Efe et al., 1992]. The Star Graph (SG) has been widely used as an alternative to hypercube. The important features of SG are fault tolerance, partition ability and easy routing and broadcasting. The major drawback of SG is increasing complexity with higher value to the next higher dimension. For the improvement of SG, another variation is introduced called Star Cube (SC). When compared with SG, the growth of SC is comparatively small and smallest possible structure contains twenty four nodes with node degree equal to four. The SC is edges-symmetric, maximally fault tolerant and cost effective [Tripathy et al., 2004]. The Star Crossed Cube (SCQ) is derived from product of SG and CC. It is very suitable for variable node size architectures. The Diameter of a SCQ is almost half of that of its corresponding SC. However, the SCQ makes no improvement in the node degree compared to the SC [Adhikari and Tripathy, 2014]. Motivated from the above discussion, a new multiprocessor network named as Linear Crossed Cube (LCQ) network has been proposed which exhibits the desirable properties of similar topological networks.

In this chapter, an analysis of the said network, its various properties and a brief comparison with other existing network such as hypercube, crossed cube, star crossed cube and LEC has been given in tabular form.

4.2 NETWORK MODEL

An effective interconnection network is an integral part of any massively parallel system. The network could be modeled as undirected or directed graphs. The nodes (vertices) of such a graph represent the processing elements whereas edges (arcs) denote the bidirectional

communication channels. The length of path is known as the total number of edges that comes across in a path between two nodes. The diameter of a network is also known as the largest distance between two nodes. The Low diameter is always better because the diameter puts a lower bound on the complexity of parallel algorithms requiring communication between arbitrary pairs of nodes. The largest degree of all nodes in the network is defined as the degree of a network. The complexity of the network is determined by the connectivity of the nodes. The higher complexity means larger number of links in the network. Extensibility is the property which enables to construct large-sized systems out of small-sized systems with minimum changes in the configuration of a node of the system [Rashed et al., 2012]. The cost is evaluated as a product of degree and diameter of network. Larger the cost of a network means larger number of nodes it is having although functioning a parallel application onto it. It is the most appropriate criterion to evaluate the hardware cost as well as the overall performance of the multiprocessor network. It also provides good perception to design an economical parallel system. Interconnection topologies are evaluated in terms of small diameter, low degree, simple extensibility and low cost. The bisection width is another parameter to assess the performance of the network. It is the minimum number of edges required to be removed when a given network is cut into two halves. The high bisection width is desirable in the interconnection network [Awwad et al., 2003]. Many of these parameters make contradictory demands and therefore, a compromise is there in the design of the network. The proposed architecture is designed by improving these topological properties.

4.3 LINEAR CROSSED CUBE (LCQ) NETWORK

4.3.1 DESIGN AND ANALYSIS:

The Proposed LCQ network is an undirected graph and grows linearly in a cube like pattern. The network itself is defined through connection functions in a manner similar to that of cube connection.

Let r be the set of designated processor of R thus,

$$r = \{P_i\},\ 0 \leq i \leq N\text{-}1$$

The Link functions L_1 and L_2 define the mapping from r to R as:

$$L_1(P_i) = P_{(i+2)} \text{ Mod } N;\ P_i \text{ in } r \qquad \text{..... (1)}$$

$$L_2 (P_i) = P_{(i+3)} \text{ Mod } N \quad \text{..... (2)}$$

The two functions L_1 and L_2 indicate the links between various processors in the network.

Let Q be a set of N identical processors, represented as

$$Q= \{P_0, P_1, P_2 \ldots\ldots P_{N-1}\}$$

The total number of processors in the network is given by

$$N = \sum_{k=1}^{n} K \quad \text{...…..... (3)}$$

where, n is the depth of the network. For different depth, network having 1, 3, 5, 7….. and 2, 4, 6, 8, 10…… processors are possible. The link between different nodes in a network of 8 processors could be obtained as:

$P_0 = P_{(0+2)} \text{ Mod } N = P_2$; $P_0 = P_{(0+3)} \text{ Mod } N = P_3$

$P_1 = P_{(1+2)} \text{ Mod } N = P_3$; $P_1 = P_{(1+3)} \text{ Mod } N = P_4$

$P_2 = P_{(2+2)} \text{ Mod } N = P_4$; $P_2 = P_{(2+3)} \text{ Mod } N = P_5$

$P_3 = P_{(3+2)} \text{ Mod } N = P_5$; $P_3 = P_{(3+3)} \text{ Mod } N = P_6$

$P_4 = P_{(4+2)} \text{ Mod } N = P_6$; $P_4 = P_{(4+3)} \text{ Mod } N = P_7$

$P_5 = P_{(5+2)} \text{ Mod } N = P_7$; $P_5 = P_{(5+3)} \text{ Mod } N = P_0$

$P_6 = P_{(6+2)} \text{ Mod } N = P_0$; $P_6 = P_{(6+3)} \text{ Mod } N = P_1$

$P_7 = P_{(7+2)} \text{ Mod } N = P_1$; $P_7 = P_{(7+3)} \text{ Mod } N = P_2$

In order to define the link functions, we denote each processor in a set of K processors as P_{in}, n being the level/depth in LCQ where the processor P_i resides. The arrangement is shown in Figure 4. 1.

P_{01} P_{11}

P_{22} P_{32}

P_{43} P_{53}

……..

……..

Figure 4.1: Arrangement of Processor in LCQ network

As per the LCQ extension policy, one or two processors exist at level/depth n. Thus at level 1, P_0 and P_1 exit and similarly at level 2, P_2 and P_3 exist and so on. The LCQ network with 8 processors is shown in Figure 4.2.

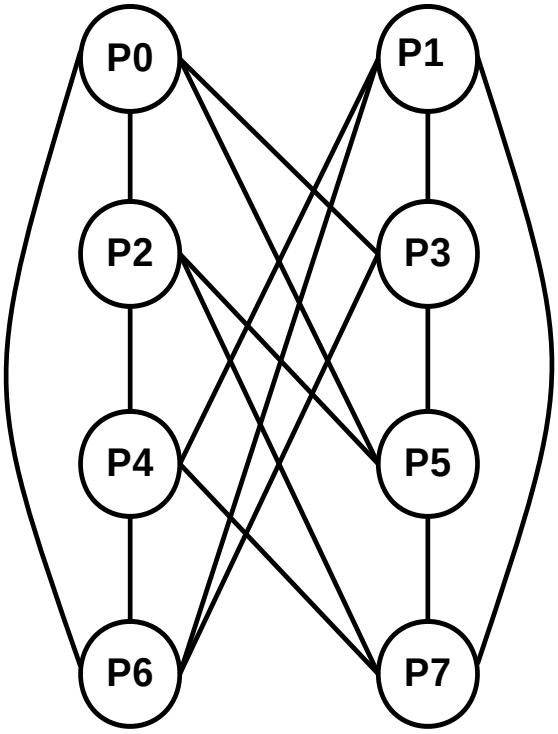


Figure 4.2: The LCQ with eight processors

4.3.2 PROPERTIES OF THE LCQ NETWORK:

In this section some salient features of the LCQ network have been compared to LEC, hypercube, crossed cube, and star crossed cube networks. These properties assist to fully grasp the capability of a certain organization. The various properties are given below:

4.3.2.1 Number of Nodes (N)

The number of nodes in a multiprocessor system performs a vital role to examine the performance of a multiprocessor network. The lesser number of nodes means complexity of system is lesser as well as the system is considered more economical. Therefore, number of nodes should be optimal. The LCQ network has number of nodes which is N = $\sum_{k=1}^{n} \boldsymbol{K}$ whereas, the number of nodes in the hypercube and crossed cube networks is 2^n.

Due to lesser number of processors in the LCQ network, it may be considered more economical than other networks.

4.3.2.2 Diameter (D)

The measure of the maximum inter-node distance in the network is known as diameter of a network. This property is important to determine the distance involved in communication which ultimately enhances the performance of multiprocessor systems. In simple words the maximum shortest path between source and destination node is called diameter of a network. The diameter of a network is bound to increase as the size grows unless there is no limit on the number of links. In case of LCQ network, it is observed that the diameter does not always increase with the addition of a layer of processors. It could be highlighted that the diameter of LCQ get an optimum value of 20 for 120 processors.

4.3.2.3 Degree (d)

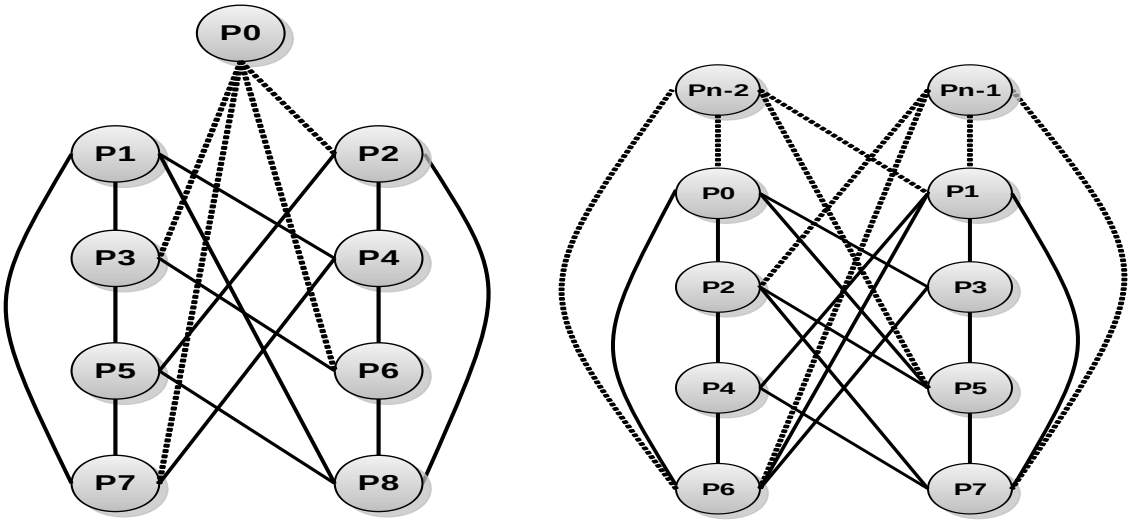
The number of connections required at each node in a network is defined as the degree or connectivity of a node. The connectivity of the nodes determines complexity of the network. The higher the connectivity, the higher the hardware complexity and hence the cost of the network. Therefore, the node degree should be kept as low as possible in order to reduce the cost. Constant node degree is better. The node degree in the proposed architecture is always 4 or less. The connectivity of LEC networks is also 4, whereas, the connectivity of hypercube increases with the size. The crossed cube is having node degree equal to (n+1)/2 and therefore connectivity in this case also increases with size. The behavior of LCQ with different parameters is demonstrated in in Table 4.1.

Table 4.1: Properties of LCQ multiprocessor network

Level	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**15**
Number of processors	0	1	3	6	10	15	21	28	36	45	55	66	120
Diameter	0	0	1	2	2	3	4	5	6	8	9	11	20
Degree	0	2	3	4	4	4	4	4	4	4	4	4	4

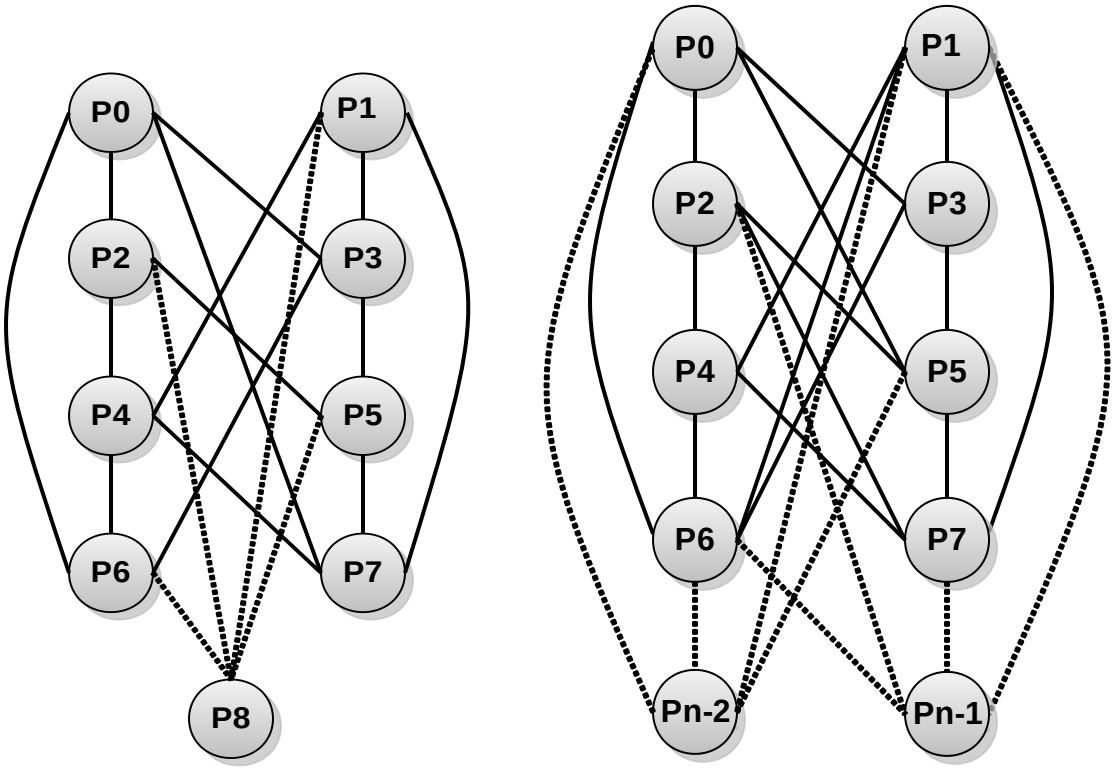
4.3.2.4 Extensibility

It is the property which enables to design large sized system out of small ones with minimum or no alteration in the configuration of system. The system can be expanded in a useful way with the help of smallest increment. The proposed LCQ network is linearly extensible which means that its extension can be carried out in a linear fashion by adding one or two nodes in every extension. When single or odd number of nodes are added into the network it is termed as odd extension. Similarly, an even extension can be performed by adding two or more even number of nodes in particular extension. The notable feature of the proposed LCQ network is that it does not have an exponential extension. If we compare the extensibility of LEC, the each extension requires single layer of 2 nodes and extension complexity increases in a constant manner. Therefore, at higher level the number of nodes becomes bulky and performance may degrade. Similarly, the hypercube, crossed cube and star crossed cube networks though are extensible but the complexity increases exponentially by the power of 2. Besides, the LCQ network can be extended in two directions, upward odd or even and downward odd or even and a chain of the network could be formed which is not available in the case of other networks. Figure 4.3 & 4.4 show the extensibility of LCQ network in both the directions respectively. The dotted lines in the figures indicate links required during expansion.



4.3 (a) Upward Odd Extension **4.3 (b)** Upward Even Extension

Figure 4.3: Upward Extension of LCQ network



4.4 (a) Downward Odd Extension **4.4 (b)** Downward Even Extension

Figure 4.4: Downward Extension of LCQ network

4.3.2.5 Cost

The node degree of a LCQ is always constant which is equal to 4, however, the diameter is $\lfloor\sqrt{N}\rfloor$. The cost could be obtained as the product of the degree and diameter. Therefore, the cost is dependent on the value of diameter.

$$\text{Cost} = \text{degree} * \text{diameter}$$

$$\text{Cost} = 4*(\lfloor\sqrt{N}\rfloor)$$

For symmetric multiprocessor network the cost factor is a widely used parameter in performance evaluation which depends upon node degree and diameter. Since LCQ has a smaller diameter in comparison to LEC network, therefore, LCQ exhibits better cost effectiveness as compared to LEC network. This trend is shown in Figure 4.5.

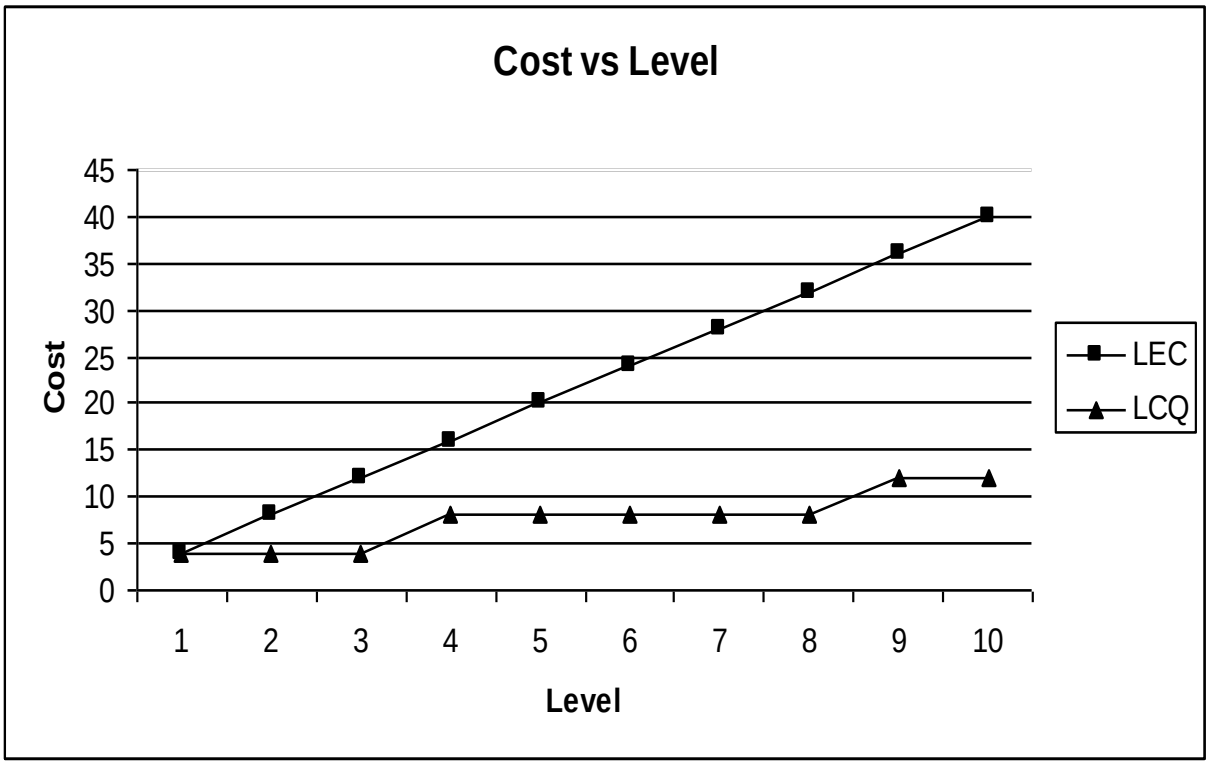


Figure 4.5: The Cost of LCQ network

4.4 COMPARATIVE STUDY

The Comparative study is made between LCQ and other multiprocessor system such as LEC, HC, CQ and SCQ in terms of important parameter of multiprocessor architectures named as diameter. The diameter represents worst case of communication delay between nodes. Figure 4.6 demonstrates the different values of diameter obtained for various multiprocessor architectures. The study of results in the curve clearly shows that the LCQ architecture has lesser diameter as compared to other similar multiprocessor networks as depicted in Figure 4.6. Considering the values of diameters in LCQ, hypercube, crossed cube, LEC and Star crossed cube networks, it is observed that in these networks, diameter increases by one at each extension. However, LCQ has smaller diameter on each level as compared to SCQ, hypercube and crossed cube network.

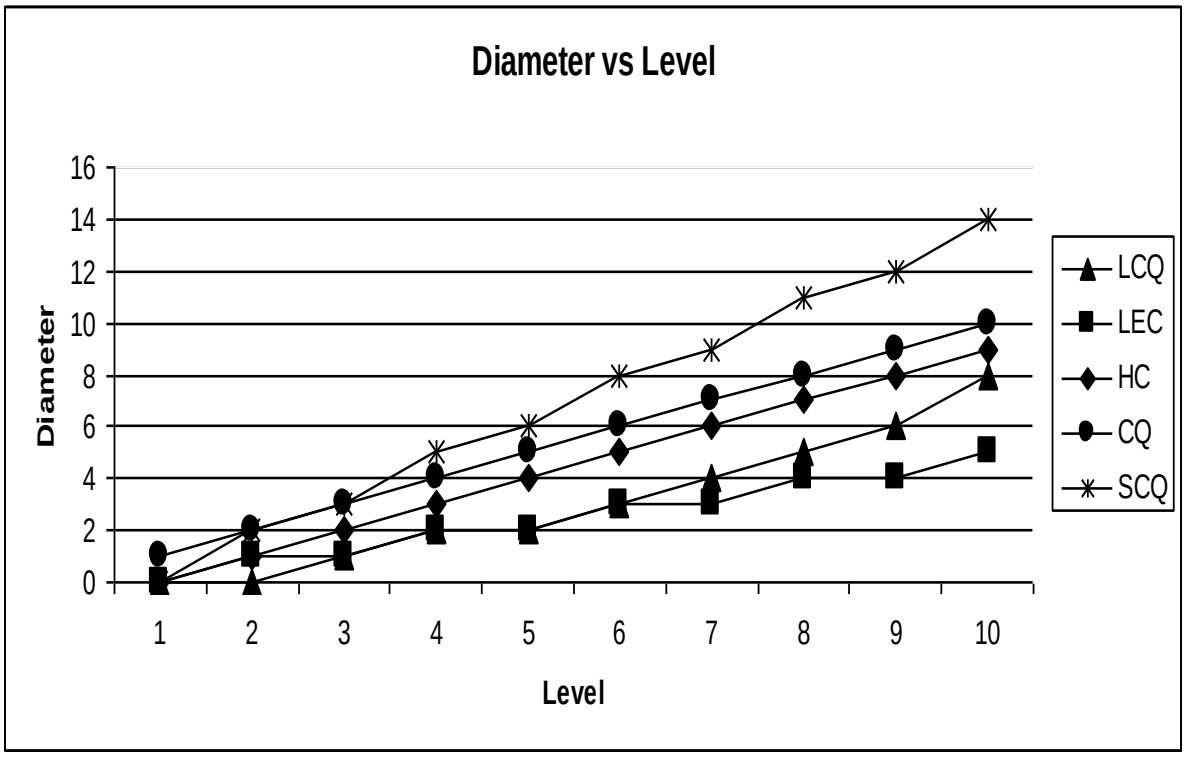


Figure 4.6: Comparison of diameter of different multiprocessor networks

Table 4.2 summarizes the comparison of the above characteristics of various processor interconnection networks.

Table 4.2: Summary of parameters for various multiprocessor networks

Parameter	HC	CQ	SC	SCQ	LEC	LCQ
Nodes	2^n	2^n	n!2m	n!2m	2n	$\sum k$
Diameter	n	n	$m+\lfloor 3(n-1)/2 \rfloor$	$(m+1)/2\rceil + \lfloor 3(n-1)/2 \rfloor$	$\lfloor n \rfloor$	$(\lfloor \sqrt{N} \rfloor)$
Degree	n	$\lceil n+1/2 \rceil$	m+n-1	m+n-1	4	4
Cost	n^2	$n \lceil (n+1)/2 \rceil$	$(m+n-1)*(m+\lfloor 3(n-1)/2 \rfloor)$	$(m+n-1)(\lceil (m+1)/2 \rceil + \lfloor 3(n-1)/2 \rfloor)$	$4 \lfloor n \rfloor$	$4(\lfloor \sqrt{N} \rfloor)$

4.5 CONCLUSION

It may be argued that a new interconnection topology (LCQ) for multiprocessor systems has been proposed as an attempt to combine the desirable features of linear architecture as well as compact hypercube or crossed cube structures. The proposed architecture exhibits lesser

number of nodes, better connectivity, lesser diameter, and linear expansion with even and odd extensions at each level over Hypercube, LEC, Crossed Cube and Star Crossed Cube networks. Therefore, the proposed LCQ architecture is a low cost multiprocessor architecture that can be used for parallel system as an interconnection network.

In the next chapter, a dynamic scheduling scheme is described which takes into account the adjacency matrix of network interconnection for migration of load on the various processors of the network. The proposed scheme is enforced on LCQ by simulating different types of load to evaluate the performance. The performance of LCQ network is compared by implementing the proposed scheme on other multiprocessor architectures discussed above.

PERFORMANCE MEASURE STRATEGIES

5.1 INTRODUCTION

The efficiency of a multiprocessor network is not merely influenced by the interconnecting pattern but also depends upon how exactly the loads are dispersed on various nodes. Scheduling of load (tasks) and resource management are consequently crucial issues while optimizing the overall performance of a multiprocessor network. The performance of a multiprocessor system is determined by the effective use of processing elements (nodes). In these systems, if certain node stays idle while others are overloaded it results significant degradation in the performance of the system. Hence, to render efficient utilization of a multiprocessor network it is necessary to distribute the load in a way which makes almost all the processors equally busy. Assignment of tasks among numerous processors in the network and optimal use of the resources typically known as task scheduling [Teodorescu and Torrellas, 2008], [Jia et al., 2010]. The primary goal of scheduling scheme is to speed up the execution of applications with efficient resource utilization especially when the workload varies at run time in an unpredictable way [Martelli et al., 2013], [Dodonov et al., 2010], [Hwang et al., 2008].

Mapping applications to parallel machine and balancing load on parallel processors is probably a complex task and continues to be resolved as research issue. A number of load balancing algorithms have been suggested and implemented on extremely parallel computers [Lan et al., 2002], [Guan et al., 2011]. All these algorithms are categorized in a variety of ways such as static/dynamic, global/local, sender/receiver and/or synchronous/asynchronous. These varieties have been discussed in detail in Chapter 3.

The crucial purpose is that in a multiprocessor system in which size of the load changes regularly as well as in an erratic method, a dynamic approach gives much better performance [Jin et al., 2011], [Zoghdy et al., 2012]. A dynamic algorithm allocates/reallocates load at runtime without priory task information which can figure out when and which tasks may be migrated [DeMello et al., 2006], [DeMello et al., 2007]. An epic dynamic scheduling scheme namely Minimum Distance Scheduling (MDS) for load balancing has been reported [Samad and Rafiq, 2005]. The algorithm makes use of a minimum distance property. Minimum distance is the property which assures the minimization of the communication in distributing subtasks and collecting partial results. The Two Round Scheduling Scheme (TRS) is another popular algorithm which has been used for task scheduling in parallel systems [Samad et al., 2013].

In this chapter, first the properties of MDS and TRS algorithms are described in brief. Their merit and demerit are highlighted from implementation point of view. Their results are analysed after implementation. After evaluation and analysis of results a new dynamic algorithm named as Optimal Multi-Step Scheduling (OMSS) is proposed. This algorithm includes the merits of both the algorithms i.e. MDS and TRS. The proposed OMSS scheme has been implemented on the proposed LCQ and on other standard reported topologies.

5.2 EXISTING DYNAMIC SCHEDULING SCHEMES

The performance of a multiprocessor system can be characterized by communication delay, distribution of load among the processors and scheduling overhead. The MDS scheme which is based on the principle of minimum distance property minimizes overhead and ensures the maximum possible speedup [Kang et al., 2011]. In the MDS scheme, the adjacency matrix of a network is used to satisfy the minimum distance property. A ‘one’ in the matrix indicates a link between two nodes whereas a ‘zero’ indicates there is no link between nodes. For load balancing, the MDS algorithm calculates the value of Ideal Load (IL) at various stages of the load (task generation). The IL is the load a processor is having when the network is fully balanced. The processors having a load value greater than the IL are considered as overloaded processors. Similarly, processors having lesser load than the value of IL are termed as underloaded processors. In other words the overloaded (donors) and underloaded (acceptors) processors are identified based on a threshold value known as IL. Each donor processor, during balancing selects tasks for migration to the various connected and underloaded

processors (i.e. the processors having a 'one' in the adjacency matrix) and thus maintaining minimum distance. Mostly any load balancing algorithm considers the overall load on the network. However, in this algorithm the load is mapped through various stages of the task structure. Each stage represents a particular state of the task structure which consists of finite set of tasks. The implementation of MDS on LEC shows that the network has good load balancing capabilities [Samad et al., 2012].

A TRS scheme is designed for solving load balancing problem with unpredictable load estimates. The TRS algorithm works as an extension of MDS and named as Two Round Scheduling scheme. It is dynamic in the sense that no prior knowledge of the load is assumed. TRS scheme takes into consideration those acceptor nodes which are not connected directly to donor node. There may be more than one path between the donor and acceptor processors which require multi-hop. However, large number of hopes gives minimum load imbalance and hence LIF is smaller (i.e. less than the standard range of 40%) [Samad et al., 2012]. Therefore, the TRS algorithm has a constraint in the scheduling to consider only one processor as intermediate node between donor and acceptor nodes. To perform the load balancing, the algorithm calculates ideal load value for each iteration of the task structure in the same way as calculated in MDS scheme. However, it goes one step forward to MDS to further balance the load on a set of indirectly connected nodes. In general, it can be concluded that TRS scheme gives better results in terms of load imbalance, however, at greater overhead and hence cost. Based on this survey a new scheduling scheme is proposed which takes care of imbalance of load as well as communication overhead. The proposed scheme is described in the next section.

5.3 OPTIMAL MULTI-STEP SCHEDULING ALGORITHM (OMSS)

The basic approach in MDS is to optimize the load balancing among processors under the constraint of the need to keep message path lengths to one hop and thus satisfying the minimum distance property. Migration from donor processor is done to the directly connected acceptors. Thus, for every donor there is a set of Minimum Distance Acceptors (MDA). Tasks are not allowed to migrate to acceptors which are outside this set. To obtain MDA for LCQ its adjacency matrix is drawn in Figure 5.1 (b).

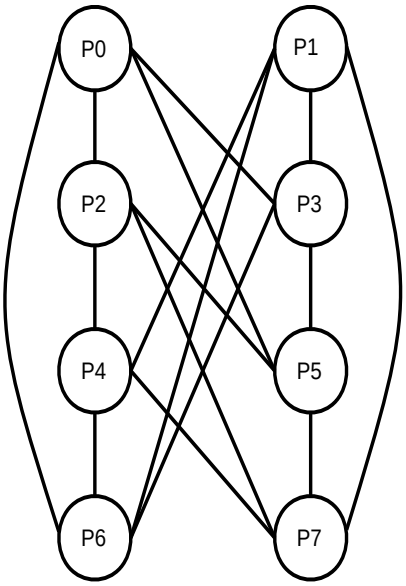


Figure 5.1 (a): The LCQ with eight processors

	P_0	P_1	P_2	P_3	P_4	P_5	P_6	P_7
P_0	0	0	1	1	0	1	1	0
P_1	0	0	0	1	1	0	1	1
P_2	1	0	0	0	1	1	0	1
P_3	1	1	0	0	0	1	1	0
P_4	0	1	1	0	0	0	1	1
P_5	1	0	1	1	0	0	0	1
P_6	1	1	0	1	1	0	0	0
P_7	0	1	1	0	1	1	0	0

Figure 5.1 (b): Adjacency matrix for LCQ network

Referring to Figure 5.1 (b) for LCQ network, MDA (P_0) = { P_2, P_3, P_5, P_6,} which indicates that even if the processor P_1 is underloaded, it would not be considered as a part of the balancing process. Therefore, a more dynamic nature of algorithm is required to make the network fully balanced which takes into consideration those processors also that are not directly connected at lesser overhead.

A new scheme has been proposed for solving load balancing problem with unpredictable load estimates. The proposed algorithm works as an extension of MDS and named as Optimal Multi-Step Scheduling Algorithm (OMSS). It is dynamic in the sense that no prior knowledge of the load is assumed. Decision of migration is taken on the fly based on the current system utilization. Adequate number of nodes with multiple paths are available that may be considered as donor and acceptor nodes. Selection of nodes for task migration is challenging and directly affects the communication cost. In order to reduce the communication cost, the OMMS Algorithm selects directly connected nodes at first step for the purpose of task migration. To avoid greater communication overhead, the algorithm does not select nodes with multiple hopes. Once directly connected processors are exhausted, the OMMS Algorithm identifies highly imbalanced nodes irrespective of their connectivity. After identification it searches those paths between these nodes which require only a single intermediate node that could be involved for task migration. The other nodes though are imbalanced, however, are not taken into consideration for task migration. For instance: processor P_1 (donor) is an overloaded processor and processors

P_4 and P_7 (acceptors) are underloaded then the connectivity by one intermediate node to P_4 and P_7 is checked. If processor P_1 has no direct connectivity and also has no one intermediate node to P_4 and P_7, whereas, P_2 has connectivity with P_1, P_4 and P_7 then P_2 will be considered as intermediate node. This strategy helps to control the communication cost and overhead on the scheduler. In order to make simulation, the tasks are generated incrementally at various stages of task structure. Each stage of the task structure (load) represents a finite number of tasks. The load imbalance factor for k^{th} iteration, denoted as LIF_k, is defined as

$$LIF_k = [\max \{load_k(P_i)\} - (ideal\_load)_k] / (ideal\_load)_k \quad \ldots\ldots\ldots\ldots (1)$$

where, $(ideal\_load)_k = [load_k(P_0) + load_k(P_1) + \ldots + load_k(P_{N-1})] / N,$ …….(2)

and $\max(load_k(P_i))$ denotes the maximum load pertaining to iteration k on a processor P_i, $0 \leq i \leq N-1$, and $Load_k(P_i)$ stands for the load on processor P_i due to k^{th} iteration. Each iteration represents a particular stage (or level) of the task structure which consists of finite number of tasks. Based on the IL value, the donor (overloaded) processors and acceptors (underloaded) processors are identified. Migration of task can take place between donor and acceptor processors only.

The scheme is defined in the following five steps:

i) Map the tasks at the root processor and calculate IL at a particular stage of the task structure (Load).

ii) Transfer the load onto various available processors in the network. Check the load of each processor to identify the donors and acceptors processors.

iii) Check the connectivity of donor and acceptors with the help of adjacency matrix and migrate tasks from donor to directly connected acceptors to make the network balanced.

iv) If no directly connected processors are left, identify highly imbalance available nodes in the network.

v) Check the connectivity of these designated nodes with one intermediate processor, if it is true, perform migration else stop.

vi) Repeat the above procedure for the next stage of task structure.

The whole algorithm is implemented in ‘Java’ language. A pseudo code of the algorithm is shown in Table 5.1.

Table 5.1: The Pseudo Code of OMSS

Algorithm OMSS ()

```
OMSS ()
{
TG[0] = 1;at 0th Processor/* TG indicates task generation at a particular stage*/
While(it_count1 <MAX_L)/*MAX_L is the maximum load at a particular load stage */
{
/* calculate IL and RIL */
IL = Calculate_IL (TG);
RIL = ceil (IL);
/* For all processors check whether the load on a particular processor is exceeding the RIL (Rounded IL). If so then migrate the load*/
For(it_count2 = 0; it_count2 < P_MAX; ++ it_count2)/* P_MAX is a total number of nodes */
{
IF (TG [it_count2] > RIL)
{
/* Migrate till load at processors become equal to or less then RIL */
While(true)
{
Migrate(it_count2)
IF(TG [it_count2] < = RIL) break;
} } }
LIF = (max (TG) – IL) / IL;/* calculate LIF */
/* Enter into the next level of the task generation (TS indicates task structure)*/
TG= TS * TG;
it_ count ++;
} }
/* Functions used by the algorithm */
Calculate_IL (X[ ])
{
Sum = 0; /* x[i] indicates load at ith processor */
For(i = 0; i < P_MAX; ++i)
Sum= sum + x[i];
Return(sum / P_MAX);
}
/* Perform migrations */
```

```
Migrate (p_number)
{
/* Get the set of connected processors to the processor for which migration is being called i.e. p_number */
For(i =0; i < P_MAX; ++i)
{
If(connected (i, p_number, level))
Temp[k++] = i;
K--;
}
/ * Get the small loaded processor number */
Small= temp [0];
For(i = 0; i < P_MAX; ++i)
If(TG [temp[i]] < TG [small])
Small= temp [j];
/* Transfer the load from p_number to the smallest loaded and connected processors */
While(TG[p_mumber]!= IL || TG[small]! = IL)
{
TG[p_number] --;
TG[small] + =1;}
 }
/* Check the under loaded processors which are not connected. If any repeat the above procedure for the next level of connectivity */
}
Max (X [ ])  /* used to find the maximum load on a processor */
{
Max = x [0];
For (i =0; i < P_MAX; ++ i)
If(x [i] >max) max = a [i];
Return (max);
}
/* The processor i with processor j. Assume the level of connectivity is given (multiple level with only single intermediate processor)*/
Intconnected (int i, int j, int level) /* returns true if processors i, j are connected */
{
If (level = = 1)
Returnadj [i] [j];
  For (int k = 0; k <no_proc; k++)
```

```
  {
If (k = = i || k = = j) continue;
    If (connected (i, k, level-1) = = 1 && connected (k, j, level-1) = = 1)
    {
        Return 1;
    }
  }
Return 0;
End of procedure
```

The above algorithm has been implemented on LCQ and other multiprocessor networks for different types of load. The simulation results are discussed in the next section.

5.4 SIMULATION RESULTS

In order to evaluate the performance of OMSS Algorithm on LCQ network as well as on other multiprocessor networks the tasks are generated and mapped on to different multiprocessor systems. These networks have been simulated to test the performance of the OMSS algorithm. The tasks are generated in random pattern and mapped onto the multiprocessor network in iterative manner. The imbalance of load is evaluated and tasks are migrated based on the value of ideal load. Tasks are running concurrently regardless of their precedence. When all the tasks are mapped and no further migration is feasible the imbalance on particular stage i.e. LIF is evaluated. The same procedure is repeated for next level of task structure and so on. In addition the balancing time for each level of task structure is also obtained which is discussed in the next section.

5.4.1 OMSS SCHEME ON LCQ NETWORK

The OMSS Algorithm is implemented and tested on the proposed multiprocessor network namely LCQ network with eight processors which is fully connected. The LCQ network consists of N= $\sum K$ nodes with a constant node degree equal to 4. It is an economical topology with lesser diameter ($\lfloor\sqrt{N}\rfloor$). The behaviour of LIF at various levels of task structures are evaluated and depicted in Figure 5.2. The figure shows that initial value of LIF starts from 20 and reaches to zero. Initially, the LIF at lesser number of tasks is higher, however, its value starts reducing when sufficiently number of tasks are available to map on the system. The results show that the algorithm produces good solutions with larger number of task. It is because the larger number of tasks are balanced efficiently on larger number of processors.

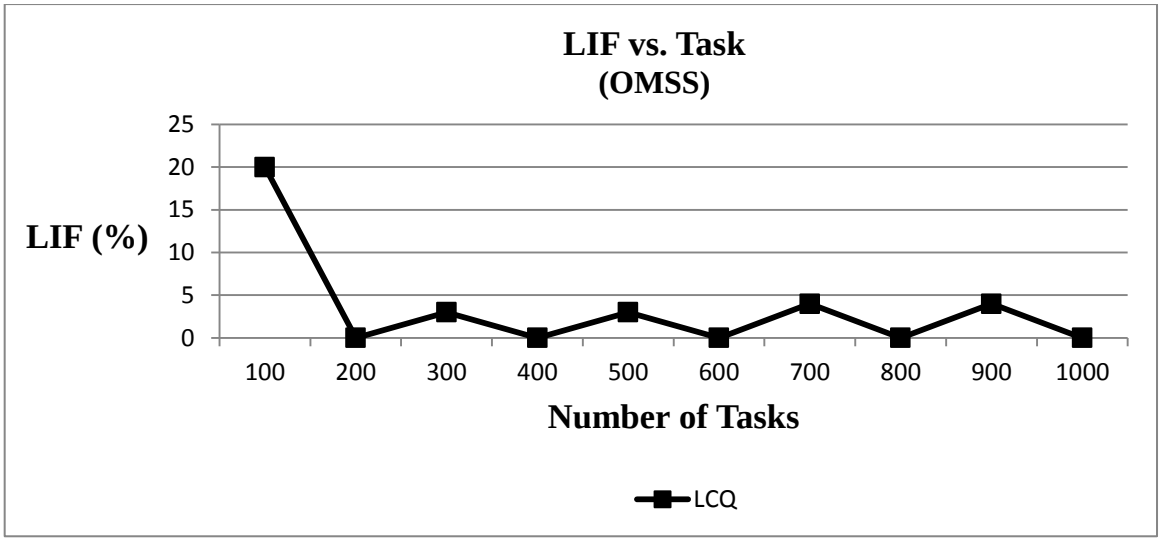


Figure 5.2: The OMSS Algorithm on LCQ Network

5.4.2 OMSS SCHEME ON OTHER MULTIPROCESSOR ARCHITECTURES

Processor topology also has significant impact on the performance of scheduling algorithms. Therefore, the same algorithm is implemented on other architectures namely Hypercube (HC) and Star Crossed Cube (SCQ) networks. In HC more processors are connected directly as compare to LCQ. When the algorithm is implemented the values of LIF are obtained at various levels of task structure. The curves are plotted as LIF against number of tasks and shown in Figure 5.3. The initial value of LIF starts from 80 and reaches to a minimum value of 20 and also then starts increasing. This behaviour is obtained due to the pattern of task structure when mapped on hypercube which is having processors as 2^n.

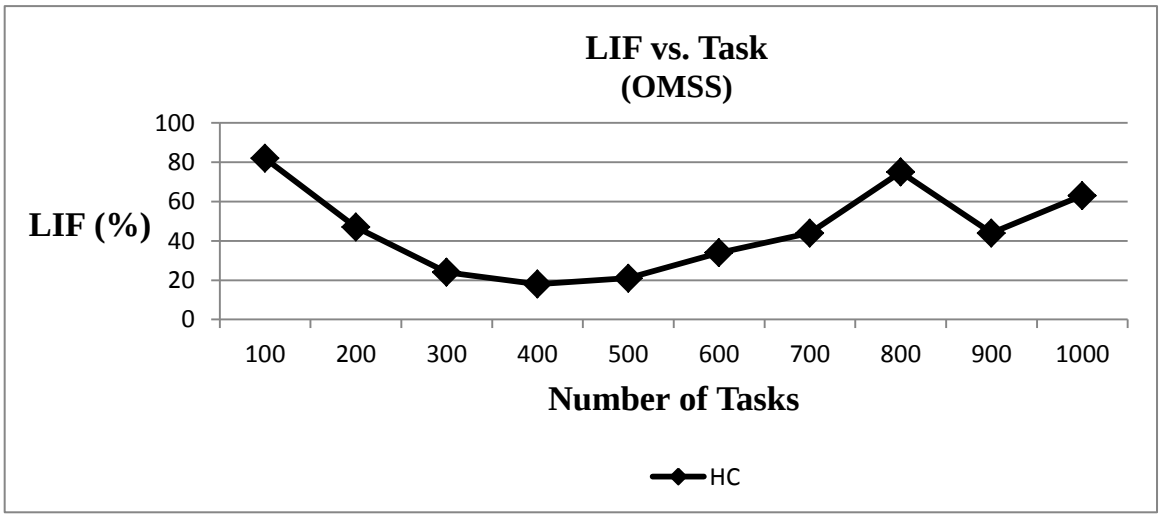


Figure 5.3: The OMSS Algorithm on HC Network

The star crossed cube is a recently reported topology [Adhikari and Tripathy, 2014] in which we considered eight processing elements for implementation and the results are evaluated with the proposed algorithm. Usually fully connected processor topology feature improves the performance of scheduling algorithm. Figure 5.4 presents the behaviour of LIF against number of tasks when the proposed OMSS scheme is implemented on SCQ. The initial value of LIF starts from 35 and reaches to a minimum value of 10. Unlike LCQ and HC, the value of LIF in SCQ starts increasing with the increase in number of tasks. The initial results are similar to hypercube architecture, however, the high values of imbalance on SCQ at later stages of task structure are obtained due to symmetrical but lesser connectivity in the system.

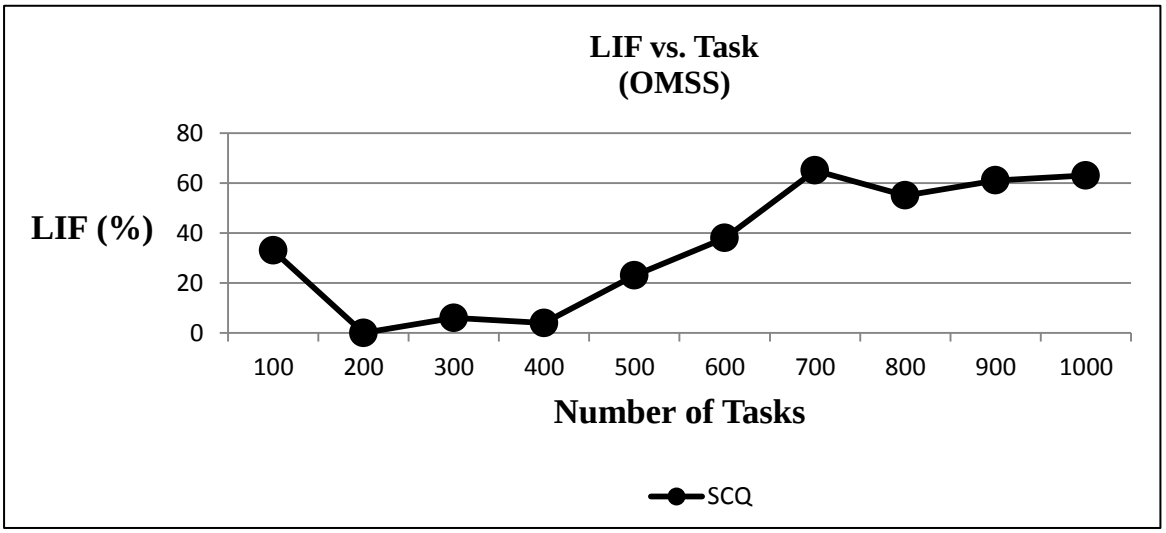


Figure 5.4: The OMSS Algorithm on SCQ Network

When comparing the simulation results it is observed that the proposed algorithm producing similar results in cube based networks i.e. HC and SCQ networks. The value of LIF is increasing at higher levels of task structures and tasks are not mapped efficiently. On the other hand when implemented on linear multiprocessor i.e. LCQ network the algorithm gives better results in term of LIF. The initial value of LIF is lesser as well as it reduces with increase in the number of tasks. This trend is depicted in Figure 5.5.

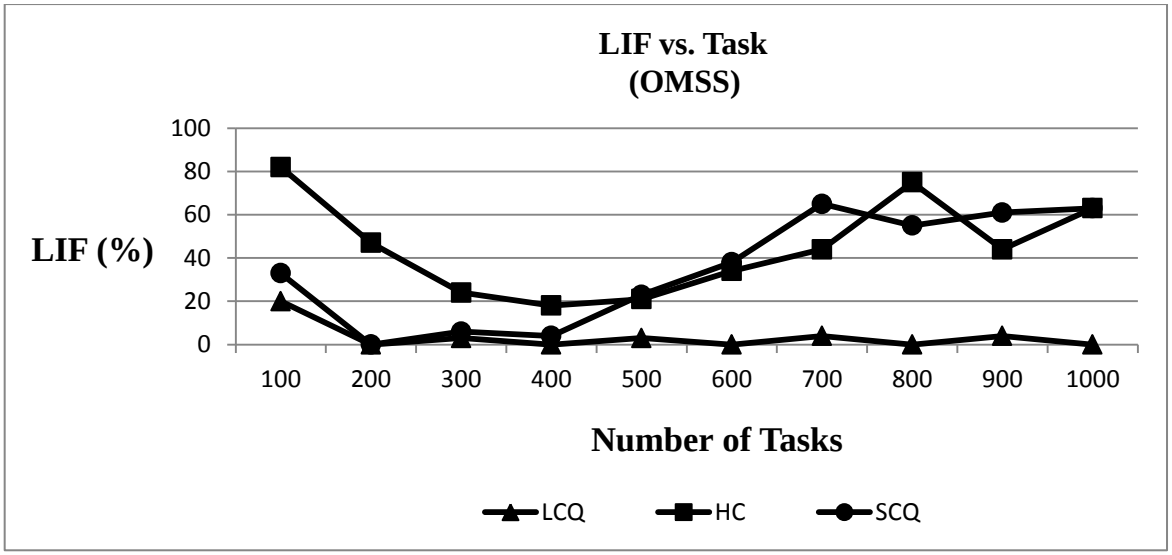


Figure 5.5: Comparison of OMSS Algorithm on various networks

5.4.3. BALANCING TIME WITH OMSS SCHEME

The balancing time is another performance parameter which is evaluated with the same algorithm when applied on different multiprocessor networks. It is observed that the proposed algorithm again producing similar results in terms of balancing time in cube based networks. The Balancing time on LCQ, HC, and SCQ networks varies when OMSS and other scheduling algorithms are applied on them. The balancing time in HC and SCQ networks is initially reducing with the increase in number of tasks; however, there is more variation in time at later stages of task structure. The balancing time is continuously smaller in case of LCQ as compared to HC and SCQ networks. This trend is depicted in Figure 5.6.

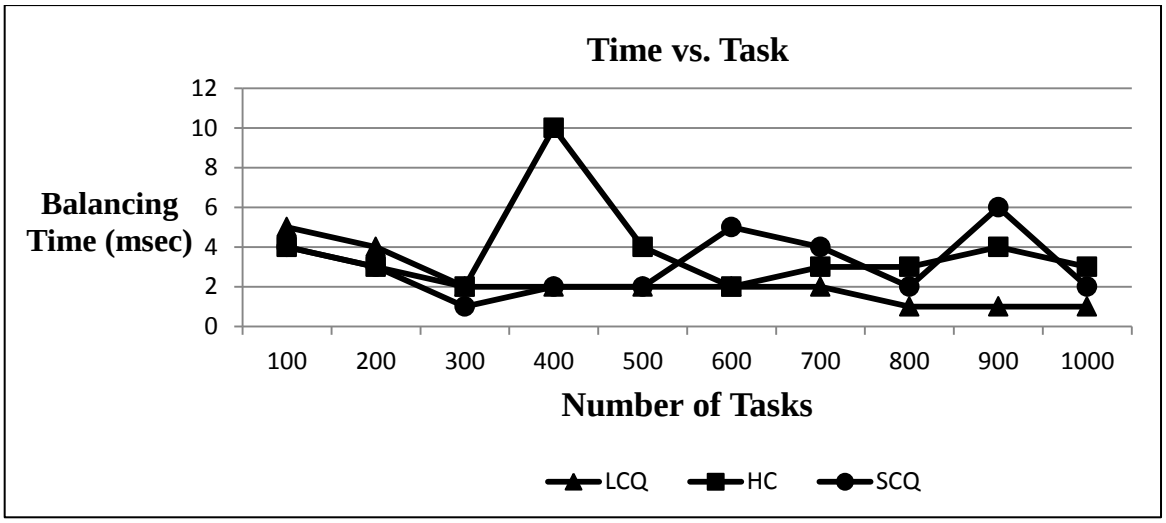


Figure 5.6: Total Execution Time of OMSS Algorithm on various networks

5.5 COMPARATIVE STUDY

To authenticate the performance of the OMSS scheme, it is desirable to implement and compare the performance of other standard dynamic scheduling schemes. The OMSS scheme and two standard reported dynamic scheduling schemes which have already been discussed are implemented on the LCQ network as well as on other multiprocessor networks under the same environment. The performance is evaluated in terms of load imbalance factor and balancing time and described in the next section.

5.5.1. PERFORMANCE OF OMSS ON LCQ NETWORK

The Minimum Distance Scheduling (MDS) and Two Round Scheduling Scheme (TRS) are considered and implemented to evaluate the performance of LCQ, HC, and SCQ networks. These Schemes were designed especially for similar multiprocessor networks particularly fully connected networks such as mesh topology. These three schemes are tested for the same set of task structures on each multiprocessor network.

When MDS and TRS schemes are implemented on the LCQ network the tasks are scheduled with purely random task structure. The mapping of task is performed at various levels of task structures and behaviour is shown in the curves given in Figure 5.7. It demonstrates a similar behaviour for all the scheduling schemes taken into consideration. The value of LIF in the beginning when lesser number of tasks are available is high, however, starts reducing with the increase in number of tasks. Comparing the results of MDS and TRS schemes, it is to be noted that the performance of LCQ with TRS scheme is better as compared to MDS scheme. However, the performance of LCQ is further improved when OMSS is implemented. In this case the LIF is continuously reducing and become zero at one thousand tasks. This behaviour is not available in other multiprocessor networks. Besides, the initial value of LIF is also lowest which also justifies the better performance than the available alternatives. A summary of results is given in Table 5. 2.

Table 5.2: Performance of LCQ Multiprocessor Networks with OMSS, MDS and TRS Schemes

Scheduling Schemes	LIF (%) (Maximum)	LIF (%) (Minimum)	Avg. Balancing Time up to 10^{th} level (msec)
MDS	186	44	4
TRS	25	6	4
OMSS	20	0	2

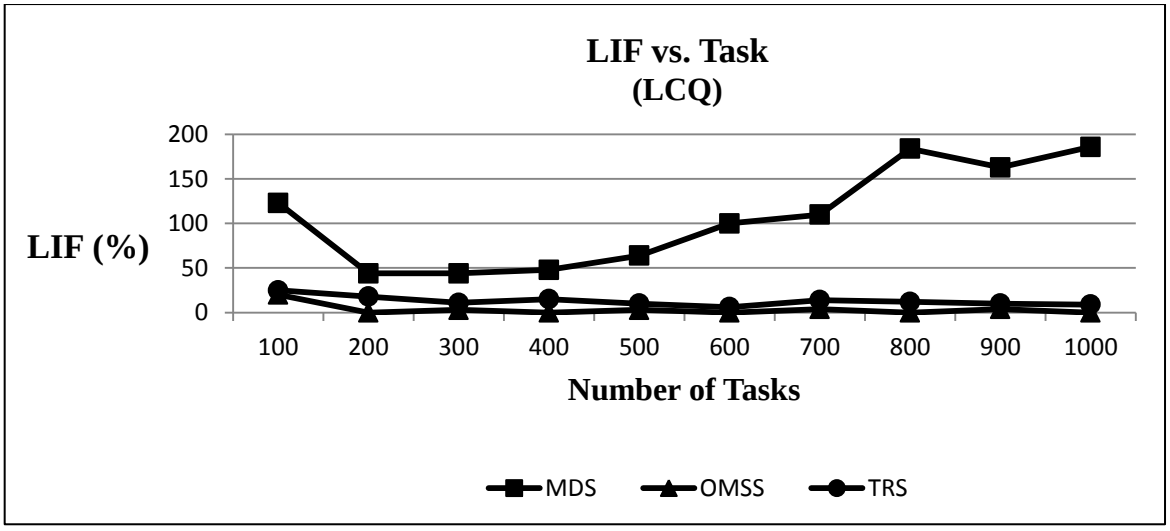


Figure 5.7: Performance of MDS, OMSS and TRS on LCQ Network

Similarly, the performance of various scheduling schemes is evaluated in terms of balancing time when implemented on LCQ. The curve is drawn as Time versus Load and depicted in Figure 5.8. It is observed from the curves that there is no regular pattern in the balancing time with load. The behaviour of the tasks is unpredictable and hence, the balancing time varies with OMSS and Other Scheduling Schemes when applied on the LCQ network. The variation in balancing time is significant when MDS and TRS schemes are implemented. On the other hand when proposed OMSS scheme is applied on LCQ the balancing time decreases and remains almost constant with increase in number of tasks. The variation in balancing time is lesser for moderate load and hence OMSS gives optimal solution for moderate load.

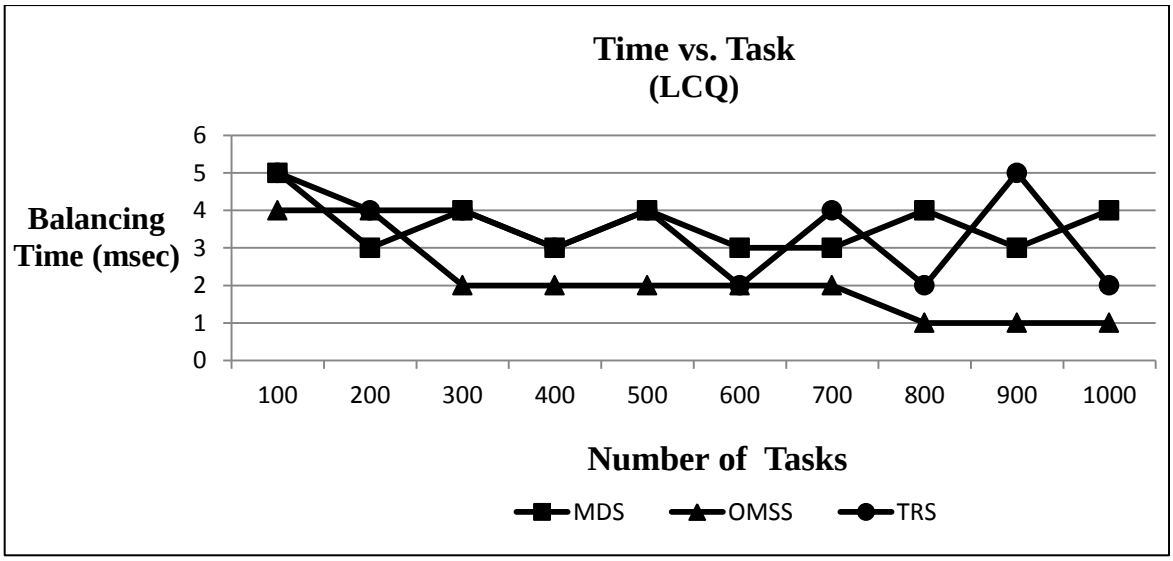


Figure 5.8: Balancing Time of MDS, OMSS and TRS on LCQ Network

5.5.2. PERFORMANCE OF OMSS ON HC NETWORK

Similar results are evaluated with the OMSS, MDS, and TRS algorithms with same task structures for HC network. The curves are plotted and shown in Figure 5.9 & 5.10 respectively. The results obtained indicate that the OMSS is performing better when implemented on the HC network. The initial value of LIF is lesser and it continously decresing as larger number of tasks are generated. The major drawback of MDS and TRS schemes is that they produce higher initial value of LIF in comparison to OMSS scheme. For intermediate stages of tasks structure i.e. for moderate load the behaviour of all the scheduling schemes is similar. On the other hand, for larger number of tasks OMSS again gives lesser values of load imbalance. Therefore, it can argue that the performance of OMSS scheme is comparatively better as compared to the MDS and TRS schemes when applied on HC network as well. A summary of results is shown in Table 5. 3.

Table 5.3: Performance of HC Multiprocessor Networks with OMSS, MDS and TRS Schemes

Scheduling Schemes	LIF (%) (Maximum)	LIF (%) (Minimum)	Avg. Balancing Time up to 10^{th} level (msec)
MDS	159	30	5
TRS	90	36	5
OMSS	82	12	2

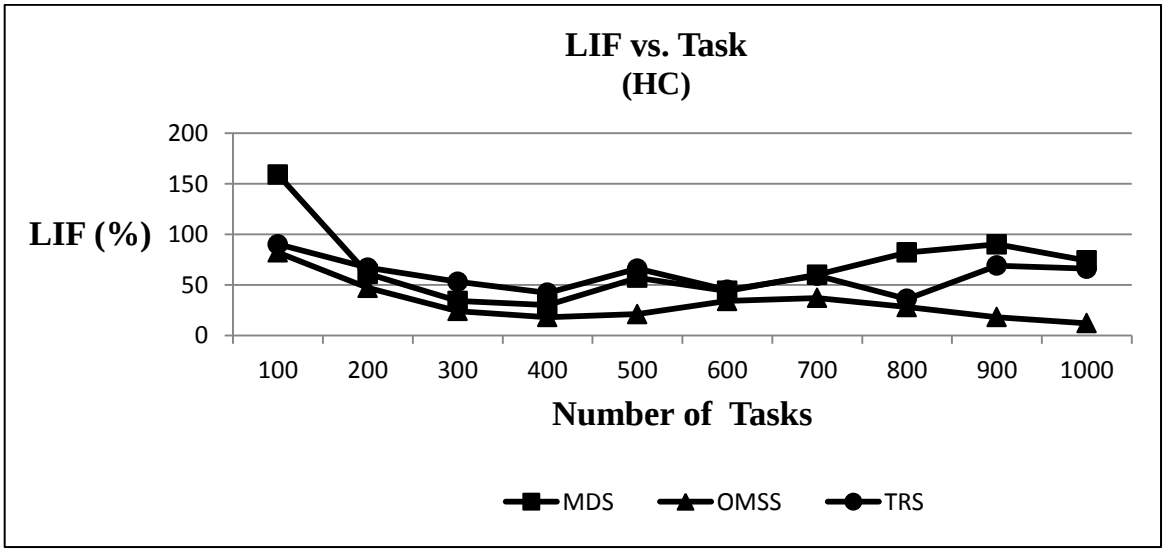


Figure 5.9: Performance of MDS, OMSS and TRS on HC Network

Figure 5.10 presents the comparison of balancing time for different networks with the three dynamic scheduling schemes. In case of HC network, the balancing time increases as the number of tasks increases when TRS Scheme is implemented. Also the MDS Scheme on HC network produces similar results in terms of time against number of tasks. However, the balancing time in HC network when OMSS is applied gives lesser value as compared to when MDS and TRS scheduling schemes are implemented on it. These results show that MDS and TRS are not performing nicely when balancing time is taken into consideration. Thus, it may be concluded that the OMSS algorithm is outperforming for HC in comparison to MDS and TRS Schemes and gives good performance in terms of LIF as well as balancing time.

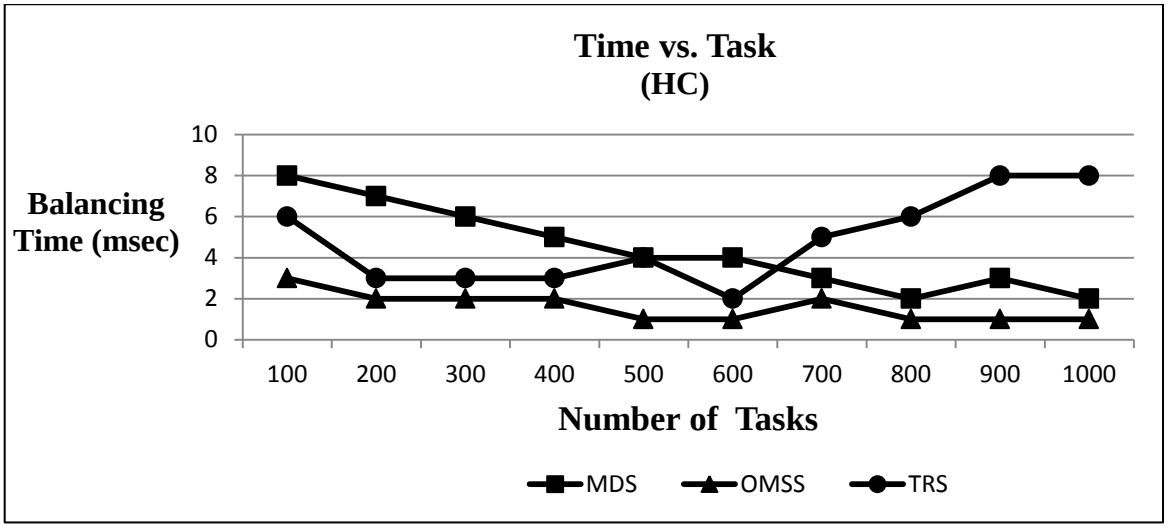


Figure 5.10: Balancing Time of MDS, OMSS and TRS on HC Network

5.5.3. PERFORMANCE OF OMSS ON SCQ NETWORK

This section presents the results of proposed OMSS scheme when implemented on SCQ network and compare them to those of MDS and TRS schemes. Figure 5.11 demonstrates the rate of decrease in LIF as the number of tasks increases. It is important to mention that the TRS and OMSS schemes are producing good solutions whereas the efficiency of MDS scheme is reduced considerably. There is a big difference between the initial values of LIF obtained by implementing three dynamic scheduling schemes. Also notable is the fact that the MDS scheme fails to make the network fully balanced for heavy load. As shown in Figure 5.11, MDS scheme is not effectively able to deal with the increase in number of tasks and the value of LIF is approaching to 236%. Therefore, an improved algorithm is required to cop up the higher numbers of task structure. The simulation

results obtained revealed that OMSS is equally performing better and is a promising approach for cube based multiprocessor networks.

Table 5.4: Performance of SCQ Multiprocessor Networks with OMSS, MDS and TRS Schemes

Scheduling Schemes	LIF (%) (Maximum)	LIF (%) (Minimum)	Avg. Balancing Time up to 10^{th} level (msec)
MDS	236	71	6
TRS	63	14	5
OMSS	55	4	3

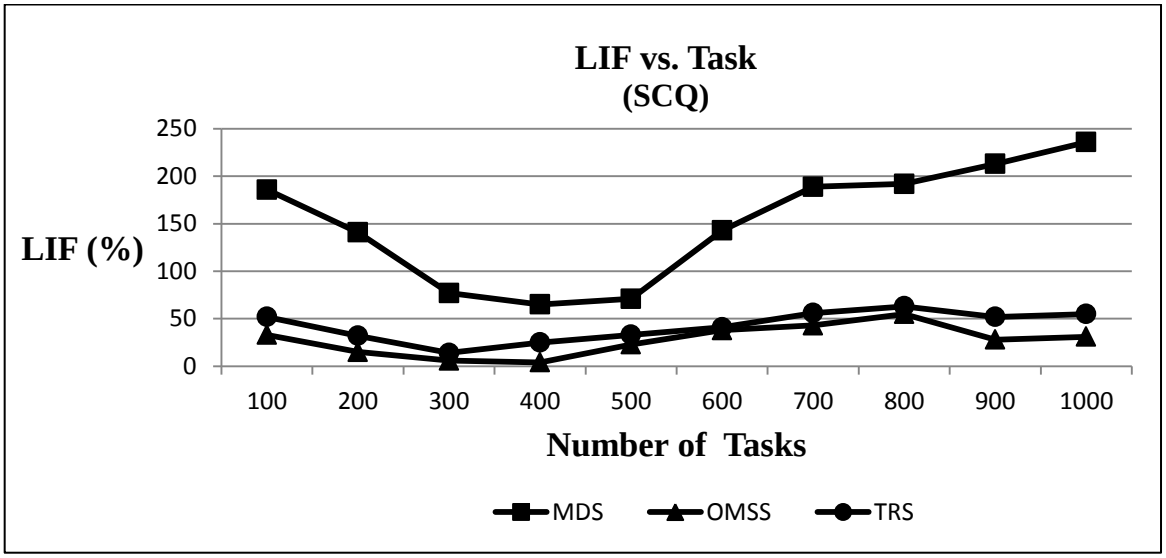


Figure 5.11: Performance of MDS, OMSS and TRS on SCQNetwork

Similarly, the performance of OMSS on SCQ network is evaluated in terms of balancing time and shown in Figure 5.12. It is observed that smaller time to balance the load is obtained by OMSS in SCQ network when tasks are larger. Therefore, the balancing time is continuously reducing on OMSS when the number of tasks is increasing. The total execution time of MDS and TRS Schemes is initially reducing with the increase in number of tasks, however, at moderate load (No of Tasks = 800) the balancing time starts increasing again when TRS Scheme is considered. On the other hand, OMSS Scheme shows the lesser balancing time throughout the increasing task structure.

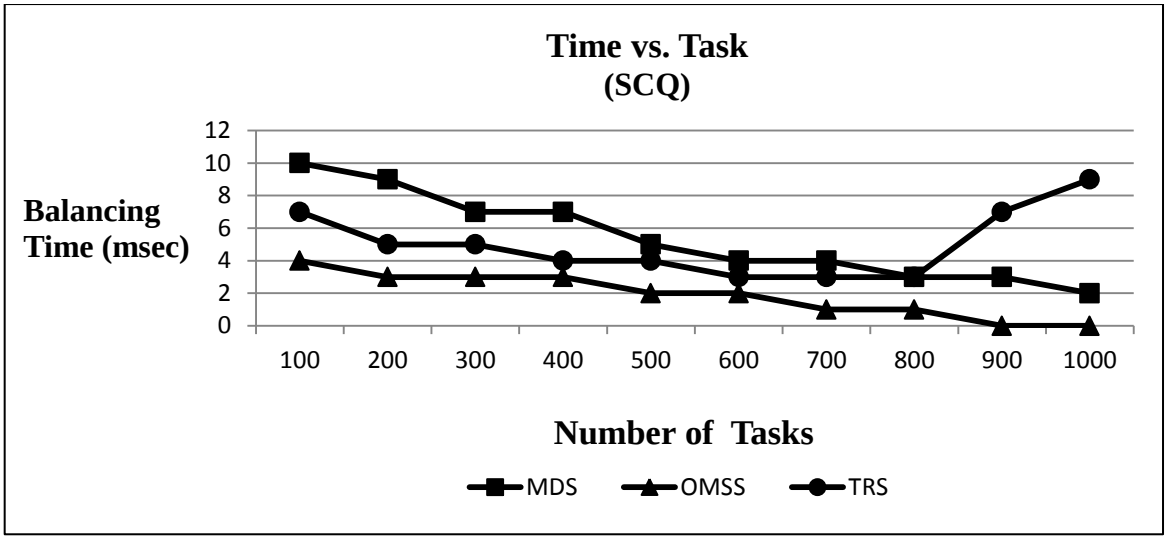


Figure 5.12: Balancing Time of MDS, OMSS and TRS on SCQ Network

5.6 CONCLUSION

The overall performance of the OMSS Scheme is highly dependent on the connectivity of the various nodes available in the network. However, the algorithm allocates the tasks to the available processors in the network whether they are connected directly and partially to indirectly connected nodes. The OMSS scheme is better, degree of balancing is higher and the network utilization is efficient and is ideally suited for linear multiprocessor networks. The proposed OMSS scheme is giving better results when implemented on LCQ network in comparison to when other scheduling schemes are applied on it. Therefore, it can be concluded that OMSS scheme and LCQ network is a good organizational model. The organization is found to be performing better particularly for unpredictable load.

In the next chapter, the eight nodes LCQ network is proposed to be used as an information retrieval server. A new algorithm for information retrieval is proposed and implemented on LCQ to test its performance for retrieval of information.

LCQ AS INFORMATION RETRIEVAL SERVER

6.1 INTRODUCTION

Information Retrieval (IR) is a process to retrieve information or data from the database through server in terms of document (files), web pages, images and text. In recent years "Information Retrieval" has emerged as a demanding tool to utilize IT infrastructure, resources and services with profound implications. Information retrieval was previously an action in which limited peoples involved for instance librarians, paralegals as well as comparable expert searchers. At present the globe revolutionized, and countless huge numbers of people involve themselves in information retrieval each day once they use a web search engine or lookup their electronic mail. Information retrieval is rapidly becoming the prominent version of information pick up, overtaking conventional database style browsing. The arena of information retrieval also deals with helping users in surfing or even blocking document collections or processing a couple of retrieved documents [Canfora and Cerulo, 2004], [Kharwar et al., 2014]. In web search, the system should bestow search over enormous amounts of documents kept on countless server computers. Information retrieval is employed today in numerous applications. It is actually a customized way to look for documents, content thereof and document metadata within conventional databases or internet. Documents could be retrieved much more easily and with smaller overhead. Retrieved documents ought to be related to a user's information requirement [Rao and Nagaraj, 2010], [Zhu et al., 2010]. Noticeable instances contain search engines as Google, Yahoo or Microsoft Live Search. Plenty of

problems in information retrieval may be thought of as a forecast issue, i.e. to forecast position ratings or rankings of internet pages, files, melodies tunes and so on, as well as learning the information desires and interests of users. However, the main objective is how to obtain economical, fast and flexible services at user's end [Zeitoun et al., 2002], [Rodriguez and Biersack, 2002].

Document Retrieval is the advanced strategy of generating a bunch of files which are pertinent to an inquirer's demand by evaluating the user's desire to a simultaneously created catalog of the textual content material of documents in the system. All these documents can then simply be found to use within the similar server [Cho et al., 2014], [Long et al., 2015]. The majority at present uses Document Retrieval systems, although they could not denote them as such, but alternatively as Web-based search engines, e.g. Google, Yahoo, Alta Vista, server etc. The three main subsystems of document based IR server are document demonstration, exemplification of users' need (queries) and the algorithms used to match queries with document illustrations [Zeng and Veeravalli, 2006], [Song et al., 2011].

The present work put emphasis on the last option. Designing high performance machines/servers is an important and effective way for enhancing the performance of Internet based services. To improve the efficiency of single node server various hardware as well as software strategies are available. In software approach a high throughput can be obtained by using bigger information cache and effective cache auxiliary methods [Cardellini et al., 2002], [Shen and Xu, 2009]. Nowadays, when hardware cost is continuously decreasing, additional computing power can be obtained by increasing the number of processing cores on a single chip. One approach to design such machines is to incorporate the concept of multiprocessor architectures [Foglia et al., 2000]. These servers can utilize the multiprocessing capability having n-processors under a single domain and hence provide cost-effective solution. This strategy meritoriously used in profitable servers such as e-Bay and Google [Barroso et al., 2003]. Another possible solution is to use a grid of clusters connected through bandwidth links. However, the overall performance shall depend on server load as well as on network latencies [Ranjan and Knightly, 2008]. On the other hand, software development paradigm has been struggling to utilize maximum performance by improving the sequential nature of most computer programs. Therefore, the programming paradigm also taking a shift towards parallelism in order to take significant benefits from these systems.

Similarly, an adaptive parallel downloading technique that can deal with the change of server efficiency has been revealed. This technique breaks a document into a number of equal items and also every block is requested from another server in parallel. The subsequent block is allocated whenever a server coatings an accessing procedure as well as gets best. Having more compact blocks results more flexibility to improve the performance of the system. However, accessing time is usually huge while the round trip time between servers along with a client is comparatively large. There is also an issue of appropriate block dimensions that will be modified based on the temporal network condition.

In this chapter, the scope of using the proposed LCQ architecture as a server for information retrieval is discussed. The LCQ has been tested by considering different set of nodes to monitor its performance. A new algorithm for retrieving information has been proposed and implemented on the proposed server. The proposed algorithm has been designed by incorporating some features of adaptive parallel downloading. The modification is made to cope up the function in user demand and capacity without acquiring additional hardware and software. This algorithm is discussed in section 6.4. A comparative study has been carried out to evaluate the performance of LCQ server with different set of processing elements. Through simulation experiments, it is shown that the resource download time is reduced considerably.

6.2 GENERAL APPLICATIONS OF INFORMATION RETRIEVAL

Internet is huge and abundant content materials are available to countless users. Consequently, it is extremely hard to examine and categorize the entire information offered on Internet. Even though search engines like Google have resolved certain issues of information retrieval from Internet. Nevertheless, extra issues are shown in search engine, for instance information overloading which minimizes the proficiency. In case the retrieval of information is massive, it brings about the holdup in downloading the information. In recent years numerous applications are available for information retrieval. Some notable applications are discussed in the fallowing section.

6.2.1 DIGITAL LIBRARY

A computerized catalogue is a library wherein selections are kept in electronic codecs (instead of print, microform, or additional media) and also available by computing

devices. The digital information might be kept domestically and found remotely through computer networks. A digital library is a sort of information retrieval system.

Numerous educational libraries are included to create institutional repositories of the institution's publications. Documents in these repositories are digitized or even "born digital" [Liddy et al., 2005], [Jarvelin and Kekalainen, 2000]. A number of these repositories are created and easily accessible to the public often having some limitations. Sometimes they are kept open access as opposed to the book of study in commercial journals in which the publishers frequently limit access rights.

6.2.2 RECOMMENDER SYSTEMS

The unambiguous kind of information blocking and recommender system that tries to recommend information items such as films, video on mandate, music, books, news, images, and web pages having high curiosity among users. Usually, a recommender system reveals a user profile to certain reference features as well as seeks to predict the score that a user would give to a product they had not yet regarded. The content-based tactic should produce the information item or even the user's social environment (the collaborative filtering tactic) [Sugiyama et al., 2004], [Costa and Roda, 2011], [Schafer et al., 2007].

Depending on the "Word of Mouth" trend that recommends things like-minded people preferred previously. However, collaborative filtering is a superb approach to ease information overload and continues to be extensively implemented in e-commerce websites. The every user choice information is not insignificant because it may increase serious issues regarding the comfort of individuals.

6.2.3 SEARCH ENGINES

Search Engine is one of the applications of information retrieval techniques to a large scale text collection. The numerous web search engines have been offered such as desktop search, enterprise search, federated search, mobile search, and social search [Fette et al., 2007], [Sugiyama et al., 2004].

To search information on the World Wide Web many web search engines have been designed. The information could include website pages, images and other sorts of data files. Certain search engines search information in data obtainable in databases or even open directory sites. In contrast to Web directories which are preserved by person editors,

search engines are managed algorithmically using human input. A worthiness opinion is a significant concern of data retrieval within web searching. Dependability of data is a pre-requisite to obtain information from study discovered onto the internet. A regularly experienced problem is that often keywords are ambiguous therefore files from a dissimilar non-relevant context are difficult to retrieve. Sometimes users are not familiar which conditions explain their problem correctly, particularly, if they are non-expert users during this domain. The unique concept of significance opinions enable users to rate retrieved documents as pertinent or much less appropriate thereby assisting additional users to find documents faster [Lv and Zhai, 2009], [Ruthven and Lalmas, 2010]. These tips when implemented for image retrieval make it easy to retrieve fast and correct graphic information.

6.2.4 MEDIA SEARCH

An image retrieval technique is an automated system for surfing, looking and also retrieving images from a sizable database of digital images. A lot of conventional as well as popular ways of image retrieval use certain procedure for attaching metadata for example captioning, key phrases, or even depictions to the images to ensure that retrieval could be carried out over the annotation terms. Guide image annotation is lengthy, repetitious, and costly. To address these issues there has been a lot of study completed on automated image annotation [Goodrum et al., 2000], [Lempel and Moran, 2003]. Furthermore, the rise in sociable internet applications along with the semantic web possess motivated the progression of a number of internet based image annotation tools.

6.3 PARALLEL INFORMATION RETRIEVAL

The huge quantities of data often have to manage by the Information retrieval systems. They have to be capable of managing a lot of gigabytes and even terabytes of documents/webpages and to build and sustain an index for these documents. To certain scope the strategies continue to be provided in literature, however, advanced data structures as well as smart optimizations by themself are not adequate any longer [Chung et al., 2004]. One computer basically lacks the computational energy or the storage features needed for indexing even a tiny portion worldwide.

6.3.1 PARALLEL QUERY PROCESSING

There are a variety of approaches wherein parallelism will help an IR server to procedure queries much faster. The two most desired strategies are index partitioning and replication [Acker et al., 2008], [Paraschos et al., 2013]. Assume that we have a complete n index server. Following the regular terminology we allude to these servers as nodes. By producing n-simulations of the index and allocating every model to a detached node, we are able to recognize an n-fold improve of the search engine's service rate without disturbing the time needed to procedure one query. This kind of parallelism is known as inter-query parallelism as numerous queries could be processed in parallel. Nevertheless every single query is prepared sequentially. On the other hand, we can separate the index into n-components and every node performs merely by itself on compact section of the index.

6.3.1.1 DOCUMENT PARTITIONING

The every index server in the document partitioned index is responsible for a subset of the documents. The consultant can obtained every inward user query by a frontend server which forwards to all n-index nodes. Delays to procedure the queries from the index nodes are merged with the search results and delivered to the users [Cahoon et al., 2000].

The primary benefit of the document-partitioned method is the ease to process the query. Since the index server takes care of query formulation virtually no other complexity must be presented into the low-level query working routines. Everything must be provided is the receiver server that forwards the query to the back ends. After getting the premier k-search results from each of the n-nodes they are made available to the end users. Along with forwarding queries and search results the receptionist can also sustain a cache which contains the outcomes for recently/frequently granted queries.

6.4 PARALLEL INFORMATION RETRIEVAL SERVER ARCHITECTURES

Different servers take different approaches towards enhancing their performance. Server architecture normally classified into one of the following categories [Choi et al., 2005]:

(1) Multi Process (MP)

(2) Multi-Threaded (MT)

(3) Single Process Event Driven (SPED), and

(4) Asymmetric Multi Process Event Dives (AMPED)

6.4.1 MULTI PROCESS SERVER

In the multi-process (MP) server architectures, a procedure is allocated to carry out the simple steps related to assisting a client demand consequently. The method works all the steps linked to serve a demand before it accepts a new request. Since multiple processes are being used, a lot of needs could be common at the same time. Every procedure possesses its own private address. Consequently, the primary disadvantage on this model is the trouble to discuss any worldwide data for instance cache information, address space and so on. An MP dependent server requirement extra storage to sustain the similar cache dimensions per procedure as compared to other server models. Hence, the efficiency of such kinds of servers is smaller as compared with additional models.

6.4.2 MULTI-THREADED SERVER

The Multi-Threaded (MT) model includes several kernel threads with one common address space. Threads are scheduled on a processor as well as every thread performs all the steps needed to serve a demand unbiased to extra procedure. The most important benefit of this model is threads could which reveals a lot of the process's sources for instance address space, open files, data cache etc. Particularly the information cache is common among all threads. The disadvantage of those kinds of servers is that these are not reinforced by almost all the operating systems. Discussing of information and data cache among numerous threads can lead to great synchronization overhead. The popular Apache server was initially intended as an MP model. Later on, it is altered by including the idea of MT models [Apache et al., 2003].

6.4.3 SINGLE PROCESS EVENT DRIVEN SERVER

The Single Process Event Driven (SPED) model works better for workloads when a lot of needs desire within main storage. The SPED could prevent context-switching and also synchronization overheads among threads or procedures. These depend upon non-blocking I/O operations. Nevertheless, this characteristic will not perform when it works relevant operations due to the restriction of present operating system [Pai et al., 1999]. The server depending on this model is executed by Zeus technology [Zeus et al., 2003].

6.4.4 ASYMMETRIC MULTI PROCESS EVENT DRIVEN SERVER

The Asymmetric Multi Process Event Driven (AMPED) model is an enhanced form of SPED therefore eliminates the fatigue of SPED model [Pai et al., 1999]. The servers depending on this model incorporate the occasion powered method of the occasion driven architecture with numerous helper procedures. The helper procedures are accountable to manage all the disk focused needs. The primary server merely serves the cache-hit requirements. When there is a cache avoid, the primary process onward the demand to a helper procedure and then the helper process fetches the information from the disk as well as transmits it returning to the primary procedure by Inter-Process Communication (IPC).

6.5 SYSTEM MODEL

The various servers are available as an information retrieval server to reduce the retrieval time with a single processor system but these systems are not performing better in terms of execution time. Therefore, multiprocessor architectures are being used to design a high performance information retrieval server. The overall retrieval system includes three objects such as an analyser, Indexer and graphical user interface with the database. We analyse different ways of crafting information retrieval systems that scales to huge document selections for instance the data files and also Web pages.

6.5.1 SERVER ARCHITECTURE

The server models discussed in the above section were basically proposed for single processor system. However, with the advent of System on Chip (SoC) architectures high performance server models can be designed. With the technology scaling down rapidly, it would be possible to fabricate SoCs with larger number of processors [Benini and Micheli, 2002], [Bell and Gray, 2003], [Edenfeld et al., 2004]. The motivation of the proposed LCQ server relies on this context and attempts to see how the server design can benefit from the architectural innovations [Zaki et al., 2015]. Effort is on to investigate how to exploit a multiprocessor architecture with lesser number of nodes i.e. lesser computational overhead. Figure 6.1 demonstrates a high performance Information Retrieval (IR) server with eight nodes named as LCQ server.

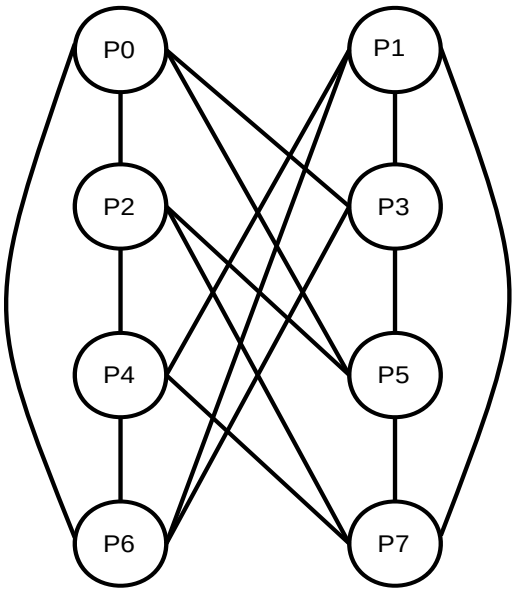


Figure 6.1: The LCQ Server

The proposed LCQ server supports a wide range of IR commands such as query, documents etc. Any node of the server can receive a client request and become the initial service node for that request. The request is served by the initial node based on the contents available at the node. Every node maintains the information related to the request, if it is available in the local database. If the request is large or the initial node is overloaded than the request is forwarded to other nodes of the system. The proposed model take maximum utilization of each node in the server without comprising parallelism. For load balancing purpose, every node shares the information related to the request, therefore, each node works as a service node.

6.5.2 PROPOSED MODEL

The relational database management system supports a stable and robust solution to handle large amount of queries while maintaining the databases. Sometimes access to data is made in a customized form which results a series of restrictions to both developer and end users. The proposed server attempts to categorize the gathered information of its database, where classification is made in such a way that each time any specific information is retrieved from the database, it is retrieved almost instantly. No any specific customization is made based on characters or bytes. For the purpose of simulation we have considered only textual file. The proposed model, however, could be extended for

other file formats. If the needed information is unavailable, a link for the identical is searched for additional linked servers.

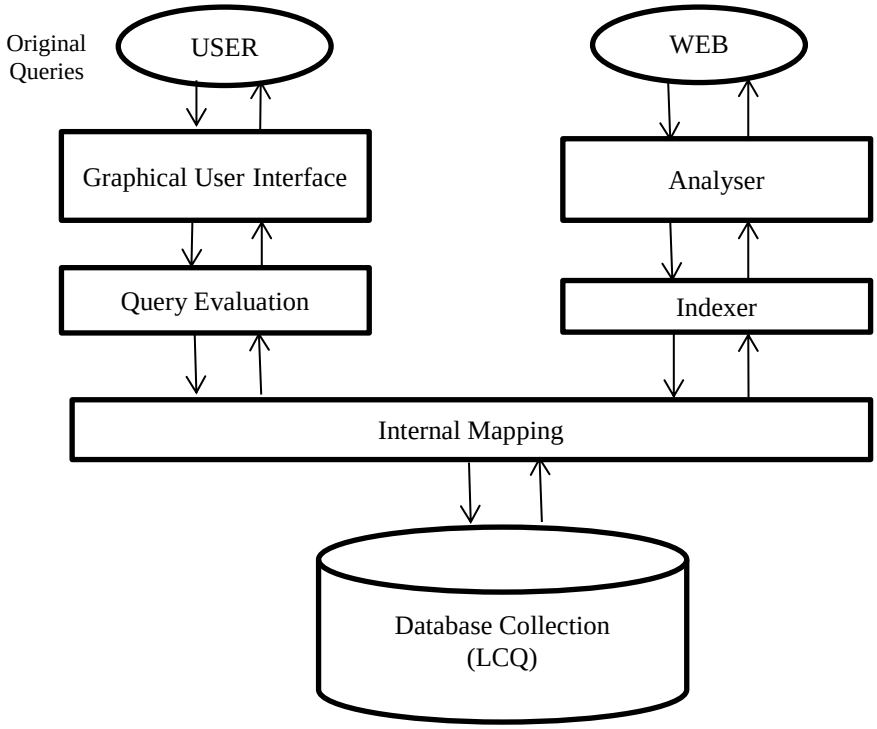


Figure 6.2: Loading and Retrieval System

In order to provide a clear concept the functionality of proposed model is divided into three major modules. These modules are termed as analyser, indexer and a user interface along with a database maintained at server site. Figure 6.2 demonstrate the functionality of each module.

An analyser is a tool that analyses given data according to the user query. It examines in detail the structure of the given data and tries to find patterns and relationships between parts of the data. The analyser works in coordination with indexer. The key role of indexer is to maintain an index of files, which is automatically generated in the background. A series of mechanism is generated in the background to exploit parallelism and improve the performance of LCQ server. In addition a user friendly tool is designed that allows users to graphically specify the information they are looking for in an easy and interactive way.

6.6 PROPOSED INFORMATION RETRIEVAL ALGORITHM

Web is a huge and copious content useful to millions of users. Therefore, it is very difficult to analyse and classify the whole information available on Web. Search engines have solved some problems of information retrieval from Internet. However, additional problems appear in search engine, such as information overloading which reduces the efficiency [Xi et al., 2000]. If the retrieval of information is large, it causes the delay in down loading the information. The proposed algorithm when implemented on LCQ server tries to classify the collected information of its database and this classification is made in such a way that whenever any information is retrieved from the database, it is retrieved almost instantly. Figure 6.3 depicts the main components of the proposed system. It consists of a web interface through which the end users can express their queries. In the background it utilizes the services of indexer maintain on the server side. The loading mechanism maintains a packet table which is being utilized for retrieval of information. When particular information is being retrieved, it is first searched in the packet table where its detail information is available. By using these details the complete information is then retrieved from the existing database directly. Users submit a query and receive result pages in a quick manner. Moreover, the model also searches the web information resource on its own initiative and traces the change of the Web information in order to automatically update and extend the local resource periodically. For those search requests that are not available in the database, it can automatically selects the commercial search engines to search information and perform classification and integration of information received from different sources.

The workflow of the proposed algorithm is as follows: The server collects information in the form of web pages or files from the internet. Each file when loaded into the database of the server is analysed and maintained through an indexer available at the server.

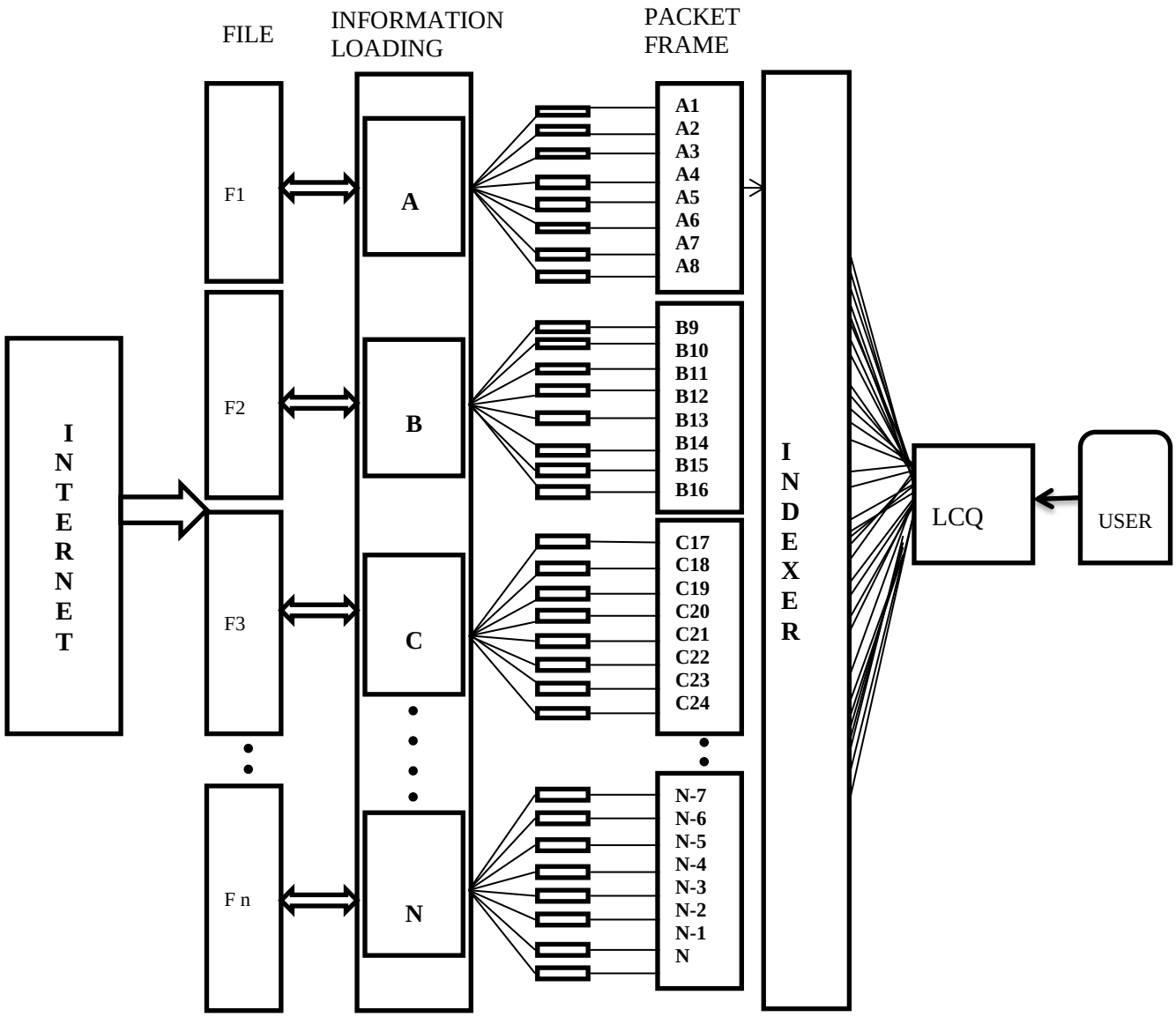


Figure 6.3: Loading and Retrieval System

The indexer is playing a role of query rewriter. In order to make meaningful and user friendly information retrieval system all the nodes of the server take part in parallel. A mapping between the loading mechanism and end users is automatically generated by the proposed algorithm. The mapping rates are placed in the correct order to execute a particular task. This mapping allows information retrieval to the end user through a graphical user interface without any coding and implementation information. A pseudo code of the whole algorithm is shown in Table 6.1.

Table 6. 1: Information Retrieval Algorithm

```
IR {}
Prepare the Document List
{
Public static void main (String args [])
 {
PrepareDocumentList Pd = New PrepareDocumentList ();
DataIndexer di = New DataIndexer ();
IF (di.conn! = Null)
di.closeSession ();
 }
Int rand = (int) (Math. Random ()*1000);
DocIdRandom dir = new DocIdRandom ();
IF (lstrnd.add (rand);
}
Return lstrnd;
 }

Merge Packet
{
Private static final ExecutorService executor = Executors.NewFixedThreadPool (limit);
Public void mergeData (List<DocumentDetails> lstdoc, DocSummary ds)
{
ExecutorCompletionService<DocumentDetails> (executor);
FileData = new byte [(int) ds.getSize () +1];
Int i=0;
FOR (DocumentDetails docDetails: lstdoc)
{
IF (docDetails! =Null) {
Final DocumentDetails dd = docDetails;
Final String ip = "10.0.17.4";
Public DocumentDetails call ()
{
HttpSendGet HttpGet = New HttpSendGet ();
DocumentDetails d = New DocumentDetails ();
```

```
Try {
d = HttpGet.sendGet (ip, dd);
 }
Return d;
}});
 }}
IF (! executor.isShutdown ())
Executor. Shutdown ();
}
Manage Indexer
{
Private Int packetSize1 = 8;
Public void createDocIndex (String docName, double size, String type, byte [] data)
 {
String sqlMaxDocID = "select max (doc_id) as max from tbl_doc_indexer";
DataIndexer di = New DataIndexer ();
Statement stmt = Null;
Try
{
Stmt = di.conn.createStatement ();
Result Set docId = Stmt.executeQuery (sqlMaxDocID);
Int dId=0;
While (docId.next ())
IF (docId==null)
DId = 1;
Else
DId = docId.getInt ("max") + 1;
Prepared Statement PS = null;
PS = di.conn.prepareStatement ("insert into tbl_doc_indexer (doc_id, }
tbl_packet_indexer (packet_data, packet_size, doc_id) Values (?,?,?)");}
Public int CheckDuplicate (String name)
 {
String sqlMaxDocID = "select doc_id from tbl_doc_indexer where doc_name=?";
DataIndexer di = new DataIndexer ();
Prepared Statement stmt = Null;
Try
```

```
{
Stmt = di.conn.prepareStatement (sqlMaxDocID);
ResultSet docId = Stmt.executeQuery ();
}
Retrieve the Document from the Indexer
 {
Public List<DocSummary> getDocumentList (List<Integer> lstRand)
{
List<DocSummary> lstDoc = new Array List<DocSummary>();
String sqlMaxDocID = "SELECT d.doc_id, d.doc_name, d.doc_size, d.type, COUNT
(p.packet_id) AS total packets FROM tbl_doc_indexer d, tbl_packet_indexer p WHERE
d.doc_id = p.doc_id and d.doc_id =? GROUP BY d.doc_id";
FOR (Integer rnd: lstRand)
{
DataIndexer di = new DataIndexer ();
Prepared Statement stmt = null;
Try
{
Stmt = di.conn.prepareStatement (sqlMaxDocID);
ResultSet docId = stmt.executeQuery ();
 Int dId=0;
While (docId.next ())
{
DocSummary ds = new DocSummary ();
ds.setDocName (docId.getString ("doc_name"));
ds.setTotalNoOfPackets (docId.getInt ("total packets"));
}
Catch (Exception ex)
 {}
 }
Return Doc;
}
```

6.6.1 LOADING OF INFORMATION LCQ SERVER

The major role in information loading is focused to maintain complex queries at run time. In the background a series of mechanism has been implemented which can later be served as information retrieval tool. One such mechanism is to use analyser and indexer. When a file is uploaded on the system the file is analysed in terms of size and the proposed algorithm hashes the file. An ID is assigned and the file is divided into several parts (or packets) of equal sizes. Each packet is again designated by its ID and loaded into the database. The addresses of packets in the database along with their IDs are recorded in a table, which is shared by all the nodes of the server and updated periodically. This table may be viewed as the indexer which indexes the data as shown in Figure 6.3. The path of the hashed file along with the packet ID is kept and maintain in the database table. For quick retrieval of information, the packets are retrieved in parallel by retrieving their ID's from the table. The algorithm is designed and shown in Table 6.2, for the above procedure and implemented on LCQ server.

Table 6.2: The Procedure for loading information

```
Loading Indexer {
        For (int i=0; i<psi;i++)
        {
        Byte [] bdata = new byte [psize];
            Int k = 0;
            For (int j=psize*i;j<psize*(i+1);j++)
            {
                Bdata [k++] = data[j];   }
            Int nsize = psize;
            If (i==psi-1)
                    {
            Int rembyte = (int) (size - psize*psi);
            For (int ij = 0; ij<psize;ij++)
            lPacket [ij] = bdata [ij];
            For (int ii =0; ii<rembyte;ii++)
                        {
                lPacket [psize+ii] = data [(int) (psize*psi+ii)];}
```

```
    If (di.conn! =null)
            Public int Check Duplicate (String name)
                {
        While (docId. Next ())
{
        If (docId==null)
                {
        Else return 0;
                }
            Return 1;
} }
```

6.6.2 RETRIEVAL OF INFORMATION

There are numerous methods for evaluation of information retrieval system such as search capability, precision, presentation of output and user effort. Generally a specified search engine is advantageous when a specific information about a subject area is needed. However, in the present work stress is made how fast the information is retrieved after a query is accepted by the proposed server. We have tried to obtain different execution results with different types of query sets. It helps to demonstrate our proposed model with better and useful results. When particular information is being retrieved, it will be first searched in the table where its detail information is available. The information available in table is shared by all the nodes of the server. By accessing these tables the complete information is then retrieved by different nodes concurrently from the existing database. Users submit a query and receive result pages almost instantaneously by accessing the table. The proposed system has been tested for processing short queries. This procedure of retrieval of information is demonstrated in Table 6.3.

Table 6.3: The Procedure for retrieval information

```
Retrieve Doc Index {
            For (Integer rnd: lstRand)
{
            DataIndexer di = new DataIndexer ();
            Prepared Statement stmt = null;
```

```
        Try     {
                Stmt = di.conn.prepareStatement (sqlMaxDocID);
            While (docId. Next ())
    {
                DocSummary ds = new DocSummary ();
                ds.setDocName (docId.getString ("doc_name"));
                ds.setTotalNoOfPackets (docId.getInt ("totalpackets"));
    }
        Return lstDoc;
     }
    Public List<Document Details> getPacketList (int id)
    {
            DataIndexer di = new DataIndexer ();
        Statement stmt = null;
        Try {
            Stmt = di.conn.createStatement ();
            PreparedStatement PS = null;
            Int dId=0;
            Int seqId =1;
            While (docId. Next ())
     {
        String sqlMaxDocID = "SELECT d.doc_id, d.doc_name, d.doc_size, d.type, COUNT
        {
        (p.packet_id) AS total packets FROM tbl_doc_indexer d, tbl_packet_indexer p WHERE
        {
        d.doc_id = p.doc_id and d.doc_id =? GROUP BY d.doc_id";

                DocumentDetails ds = new DocumentDetails ();
               ds.setPacketSize (docId.getInt ("packet_size"));
                ds.setSeqId (seqId);
    }
        Return lstDoc;
}
```

6.7 EXPERIMENTAL RESULTS AND EVALUATION

In order to understand the effectiveness of the LCQ server for the purpose of information exchange the proposed algorithm is implemented for processing a number of queries. Variety of search queries has been examined through simulation and the average times taken to process these queries are computed. These results are discussed in the next section.

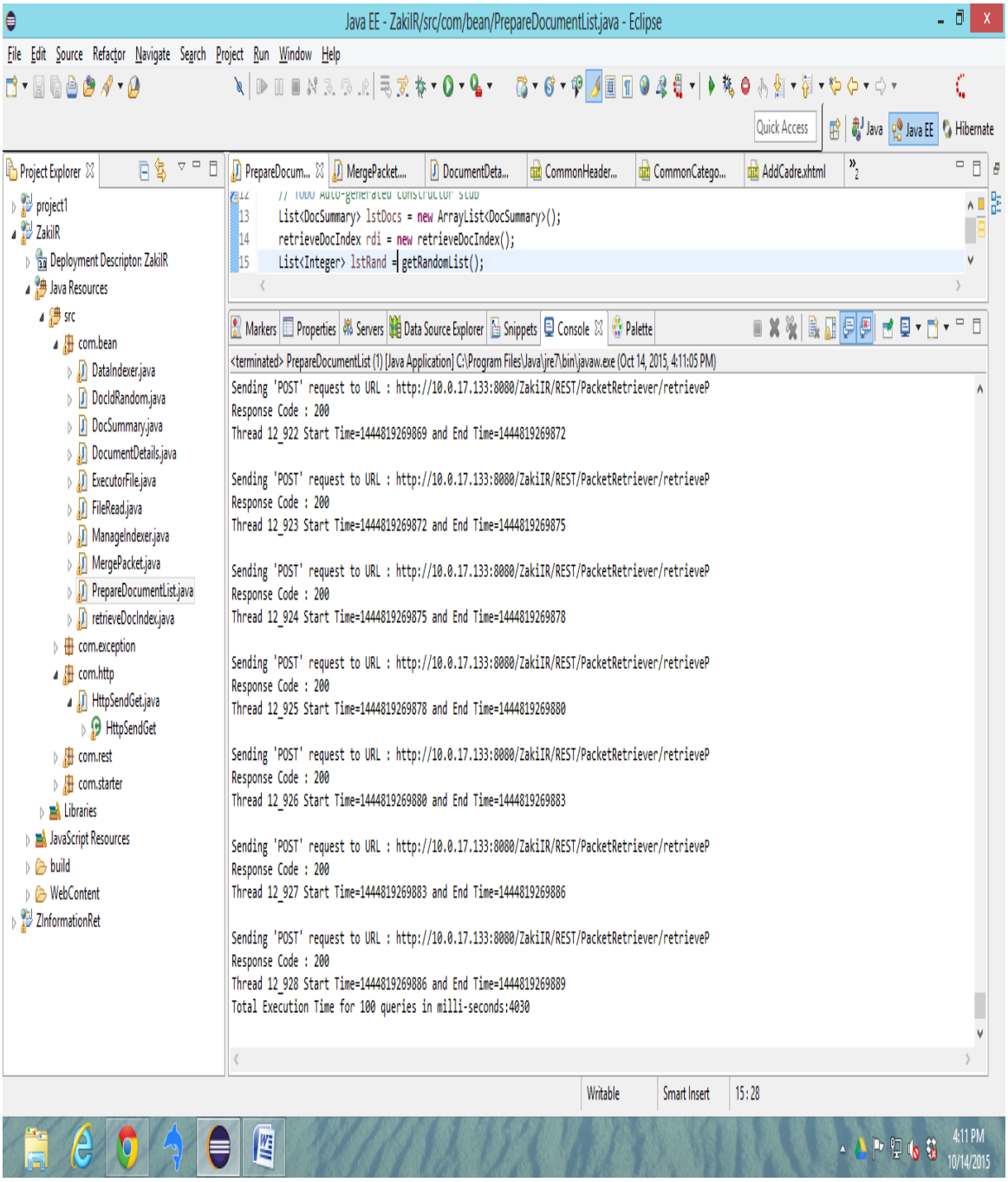


Figure 6.3: Screen Shots of IR Algorithm for queries against time

6.7.1 PERFORMANCE OF LCQ SERVER

In general the performance of LCQ server is evaluated by running simulation upon different types of queries under different systems. Besides designing the different systems we divide the queries into two different sets that are smaller to medium queries and medium to larger queries. For simplicity we considered queries between 0 to 100 as smaller to medium set of queries and queries more than 100 as medium to larger set respectively. For both the query sets the performance is evaluated on different systems as well as on ordinary system which consists of uniprocessor system. The results so obtained are shown in Figure 6.4 and Figure 6.5 respectively. These results clearly demonstrate how parallel processing can significantly improves execution performance as increase of queries directly leads to reduction of execution time. In general, the search time for ordinary server is significantly greater than the multiprocessor system (proposed LCQ server). As the number of queries increases, the ratio of searching time remains constant i.e. LCQ server is taking lesser time for searching the same number of queries in comparison to uniprocessor type server. In particular the margin between parallel and sequential system grows larger as the number of queries items climbs. The behaviour obtained in the results shown in Figure 6.5.

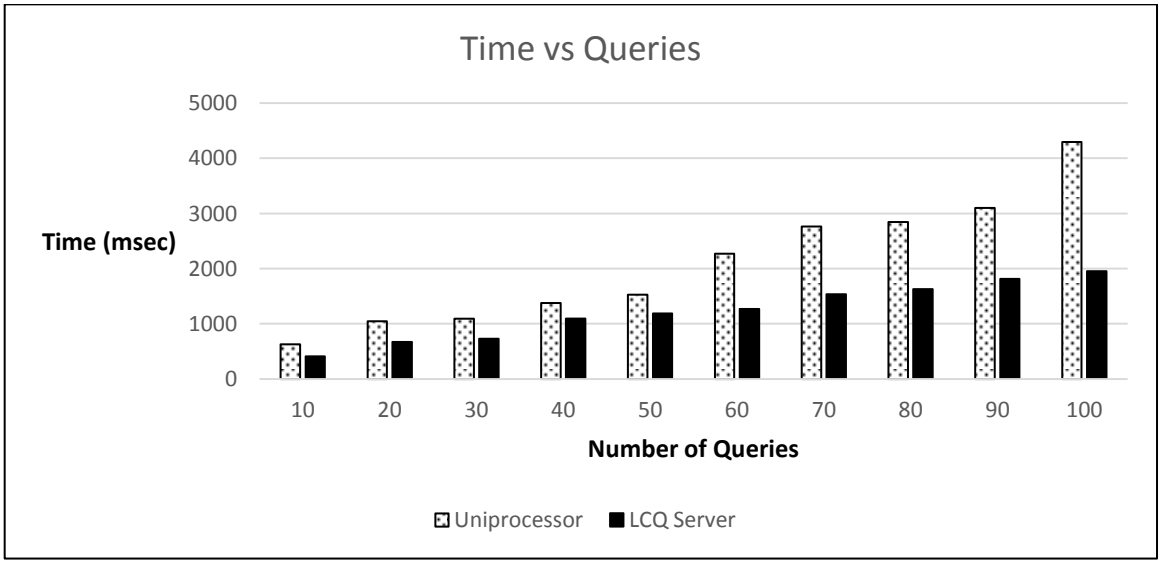


Figure 6.4: Performance of LCQ for smaller to medium queries

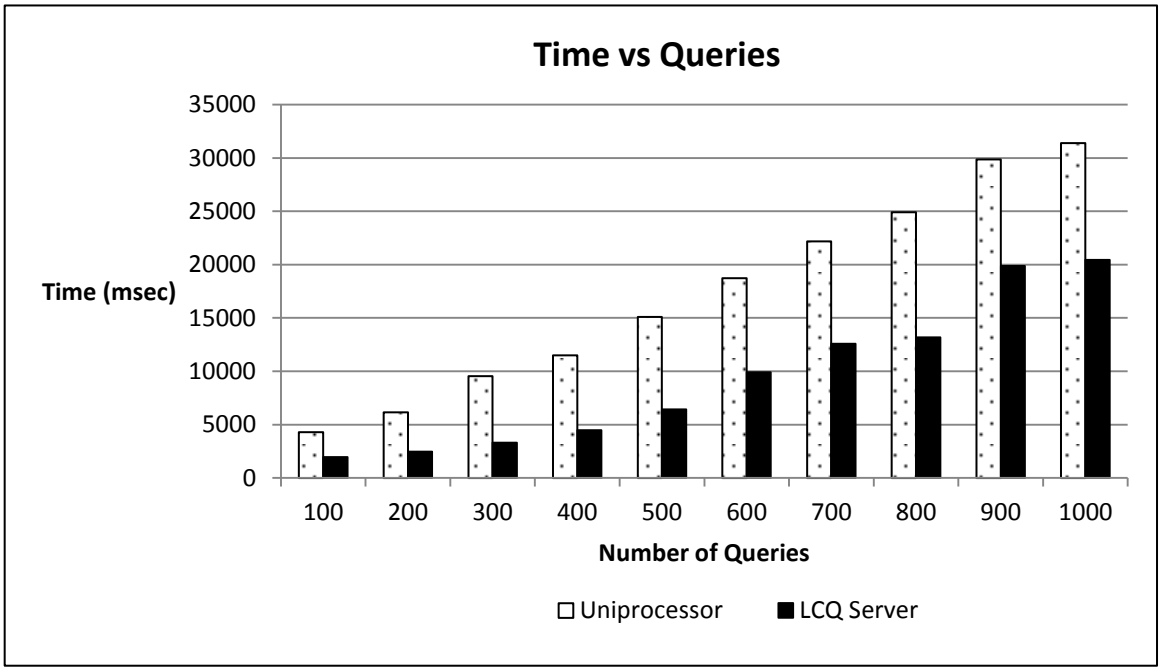


Figure 6.5: Performance of LCQ for medium to larger queries

6.7.2 COMPARATIVE STUDY

Through simulation we have worked with different systems for studying and proving the efficiency of the proposed information retrieval model. These systems are classified on the basis of number of processors used in the server. Two system means an LCQ server having four processing elements, three system means eight processing elements and four system means a server with twelve processing elements. The performances of these systems are evaluated in terms of different sets of queries namely smaller to medium and medium to larger against execution time. Working with different types of queries and their individual file, it is observed that for smaller number of queries, the behaviour of execution time is similar and has lesser variances as depicted in Figure 6.6.

On the hand, for larger number of queries we can find that the execution time for retrieving a file through different system shows larger variances. By comparing these results we can see that execution time for retrieving a file using two system is approximately 1850 mses whereas by using four system it takes approximately 1200 msec for executing the same set of queries. This indicates the improvement in execution time over different sets of processing elements. Similarly, for three system the execution time is approximately 1400 msec. From the thorough comparison of the results we can claim that the performance of proposed model is comparable for three system and four system

respectively. Therefore, to keep the overall cost lesser we can use LCQ with eight processing elements as an efficient Information retrieval server. This makes the model economical with lesser time required for execution of the whole information retrieval process using the proposed algorithm.

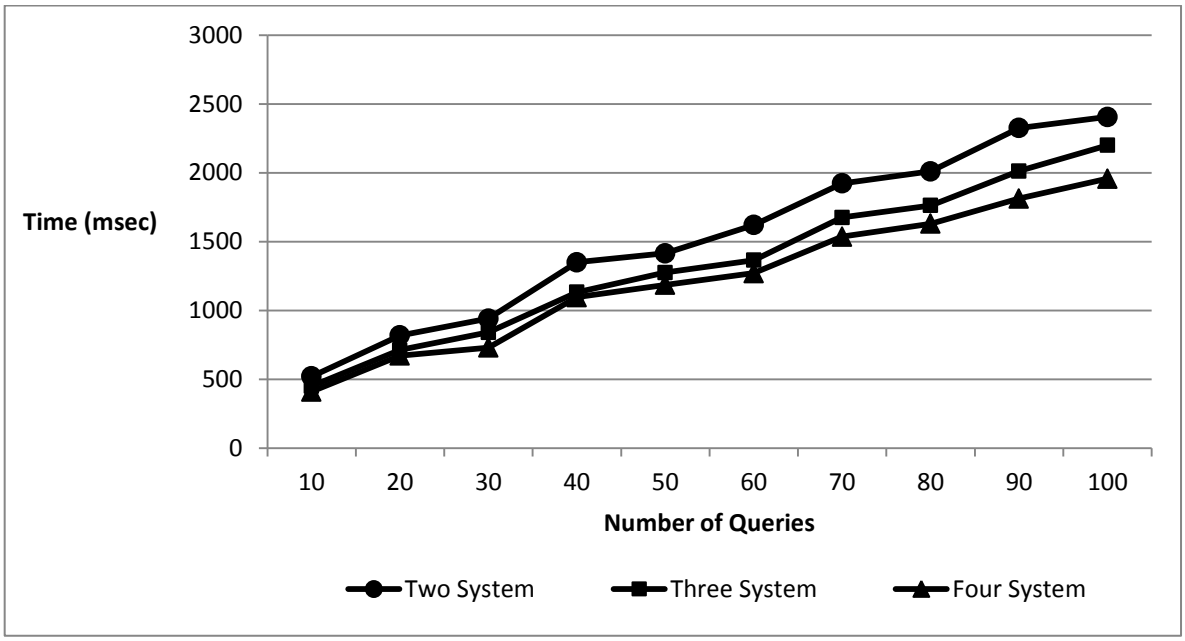


Figure 6.6: Comparison among various numbers of systems

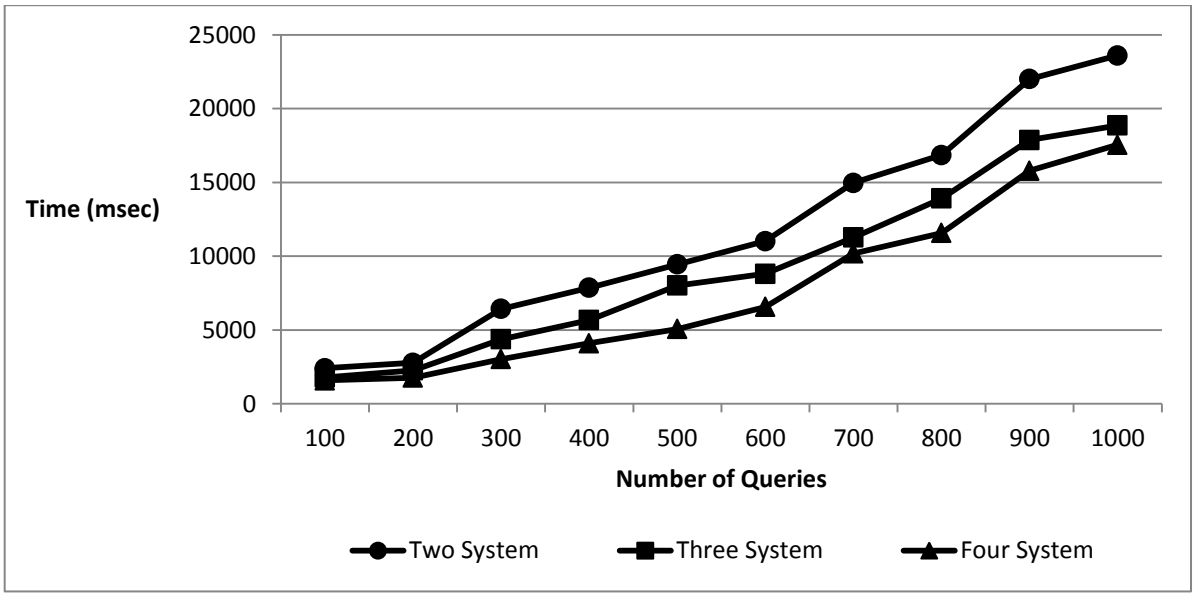


Figure 6.7: Comparison among various numbers of systems

6.8 CONCLUSION

From the above discussion it may be concluded that the proposed algorithm when implemented on the LCQ server is giving better results for larger number of queries. The given architecture with the proposed scheme is performing better for unpredictable and bursty queries and hence LCQ could be used as an information retrieval server in the networking.

CONCLUSION

7.1 INTRODUCTION

The increasing reliance on the internet as a ubiquitous medium for accessing information has compelled heavy load on the network resources. As a result, there are number of challenges in the quality of Web services. One of the desired qualities is fast accessing of information and hence small downloads time. To improve and meet the requirement of these services, efficient servers are designed. Exploiting parallelism is a necessity in the design of high performance computer system. In terms of hardware, this typically means providing multiple simultaneously active processors (nodes). In terms of software, it means structuring a program as a set of largely independent subtasks (load). Research is active in the direction of developing new multiprocessor architectures and scheduling the partitioned program onto it in order to achieve higher performance.

The work carried out in this thesis can be divided into three parts. First part deals with the design of a low cost multiprocessor architecture namely Linear Crossed Cube (LCQ) network and to study and compare its characteristics with the similar multiprocessor architectures. The second one deal with the design of a novel dynamic scheduling strategy and it's testing on the proposed network as well as on the other multiprocessor networks. Similarly, other scheduling schemes are implemented on LCQ for various types of loads in the same environment. The Comparative study shows that the proposed organization (proposed architecture and proposed scheduling scheme) is a better organizational model. Lastly, the performance of the LCQ network has been tested for the information retrieval application from the internet and has been used as a server. For accessing information a second algorithm is proposed and implemented on LCQ. The results so obtain authenticate the usability of proposed LCQ network as a server for information exchange.

7.2 CONCLUSIONS

The overall performance of the multiprocessor system is affected by a number of factors, such as communication delays, imbalance of load among the processors and scheduling overheads. The scheduling includes problem partitioning, task allocation and balancing of load among different nodes. A close correspondence between the structure of the problem and processor interconnection is desired in order to minimize these overheads. Scheduling plays an important role to improve the overall performance of a multiprocessor system. While designing a scheduling algorithm, there are many constraints, which may differ from application to application.

The basic skeleton of the proposed multiprocessor network, named as Linear Crossed Cube (LCQ), whose size grows linearly has been developed. Some of the properties observed for the LCQ network as compared to other similar architectures namely Hypercube, Cross Cube, Star Crossed Cube and LEC networks are as follows:

1) In LCQ network, the number of nodes at a level n, is N = $\sum_{k=1}^{n} K$ whereas the number of nodes in Hypercube and Cross Cube network is N= 2^n. In Star Cube as well as in Star Crossed Cube the number of nodes is equal to n!*2m. It shows that LCQ network is having lesser number of nodes and is considered more economical than other networks.
2) The degree of a node in the proposed model is always 4. The connectivity of the Hypercube is equal to the number of dimension in the cube, while in case of Star Cube and Star Crossed Cube the degree increases with size. So, constant node degree in LCQ makes the network less complex.
3) The notable feature of the proposed LCQ is the linear expansion. Each extension requires one or two nodes in LCQ. In hypercube, crossed cube and star crossed cube networks though are extensible but the complexity increases exponentially by the power of 2.
4) In LCQ network, The Cost is equal to 4ㄴn, which is significantly lesser than the other networks such as hypercube, Cross Cube, Star Cube, Star Crossed Cube and Linearly Extensible Cube.
5) The Proposed LCQ network is better fault tolerance. It has a meshed topology, and hence any single faulty link or any faulty node can be bypassed by incorporating one to two additional hopes.

A dynamic scheduling scheme known as Optimal Multi-Step Scheduling Algorithm (OMSS) has been proposed and implemented on LCQ. The performance of the LCQ network with the proposed algorithm is evaluated for solving load balancing problem with unpredictable load estimates. The behaviour of the scheduling scheme is evaluated in terms of the performance index called Load Imbalance Factor (LIF) and execution time. The LIF represents the deviation of load among processors and the balancing time shows the total time required to attain maximum value of the LIF.

The proposed scheduling scheme is also implemented on other multiprocessor networks namely Hypercube, Cross Cube, Star Cube, Star Crossed Cube and LEC in the same environment. A comparative study is carried out by drawing curves between LIF and Load (No of tasks) for different types of task structures. Similarly, the performance is evaluated in terms of balancing time with the proposed algorithm. The curves plotted from the simulation study show that the proposed OMSS scheme performs better on LCQ network for solving load balancing problem with unpredictable load estimates as compared to other networks. Another performance parameter which affects the performance of the server is the size of data, which is the candidate for migration from one node to the other. To consider the effect of block size, the simulation results are obtained for migrating load with different block sizes. It has been found that the overall performance of the network can be improved by increasing the values of block sizes. The versatility of a multiprocessor system depends upon its interconnection network and appropriate scheduling scheme. In order to confirm the performance of LCQ network along with the proposed scheduling scheme, several other dynamic scheduling schemes such as Minimum Distance Scheduling (MDS) and Two Round Scheduling Scheme (TRS) are also implemented on LCQ. Simulation results show that the proposed scheduling scheme performs better for unpredictable load estimates on the proposed network.

The second algorithm developed for managing the information exchange is applied on LCQ. The algorithm works based on the principal of optimality using analyzer, indexer and query formation. A table consisting of information of packets and their addresses is maintained which is concurrently accessible by all the nodes of the server. A number of search queries have been examined and the average times taken to process these queries are computed. A variety of information retrieval queries are executed and the execution time is evaluated. Simulation results are evaluated and compared with different types of systems. In particular, queries are executed on Multi-Process uniprocessor system as well

as on multiprocessor systems with different number of nodes. The comparative study shows that the LCQ server with eight processor outperforms with the proposed information retrieval algorithm.

Comparisons of the LCQ multiprocessor network and its inherent qualities along with lesser number of processors reveal that this network is reasonably comparable and economical with the existing multiprocessor networks. From the simulation studies, it has been found that the proposed dynamic scheduling scheme is performing better on LCQ network in comparison to other dynamic scheduling schemes. Therefore, it may be concluded that for unpredictable and bursty data the proposed architecture with the proposed schemes results in better performance. Hence, due to its various characteristics and performance parameters, the LCQ architecture could be used as server in the networking. When LCQ server is used to compare the retrieval of the information with a uniprocessor server, it is found that the LCQ type server reduces the resource download time when proposed information retrieval algorithm is applied.

7.3 FUTURE WORK

The following extensions are recommended to the work presented in the thesis.

1) The LCQ network and the scheduler may be implemented for real life problems, the actual program traces may be obtained in evaluating the performance of the network scheduling scheme.
2) The VLSI layout of the LCQ network can be studied and could be implemented using FPGA.
3) Signal Processing systems have become very complex, particularly streaming nature of signal processing algorithms could not be applied on the single node platform. These requirement tend to lead us to multiprocessing system on chips (MPSoCS). With the design of appropriate software, the proposed LCQ network could be used as a platform to map signal processing applications.
4) By applying the LCQ network in the design of a grid computing and also in the cloud computing, it could be used for various applications.

REFERENCES

[Gordon, Thies and Amarasinghe, 2006] Gordon, M. I., Thies, W., and Amarasinghe, S. (2006, October). Exploiting coarse-grained task, data, and pipeline parallelism in stream programs. In *ACM SIGOPS Operating Systems Review* (Vol. 40, No. 5, PP. 151-162).

[Pancake et al., 1996] Pancake, C. M. (1996). Is parallelism for you? Computational Science & Engineering, IEEE, 3(2), 18-37.

[Balram, Belo and Moura, 1988] Balram, N., Belo, C., and Moura, J.M.F. (1988). Parallel processing on supercomputers: a set of computational experiments. In Proceedings of ACM/IEEE International Conference on Supercomputing, PP. 247-257.

[Tiwari, Sharma and Mehta, 2015] Tiwari, R., Sharma, M., and Mehta, K. K. (2015). Dynamic Load Balancing in Parallel Processing Using MPI Environment to Improve System Performance. International Journal of Advanced Research in Computer Science and Software Engineering, Vol. 5, No. 6, PP. 730-743, 2015.

[Gkantsidis, Ammar and Zegura, 2003] Gkantsidis, C., Ammar, M., & Zegura, E. (2003, June). On the effect of large-scale deployment of parallel downloading. In Proceedings the third IEEE Workshop on Internet Applications. PP. 79-89.

[Philopoulos and Maheswaran, 2001] Philopoulos, S., & Maheswaran, M. (2001, August). Experimental Study of Parallel Downloading Schemes for Internet Mirror Sites. In Proceeding of Thirteenth IASTED International Conference on Parallel and Distributed Computing Systems (PDCS'01), PP. 44-48.

[Nor, Razi, Omar and Tahir, 2013] Nor, S. A., Razi, M. H., Omar, M. H., & Tahir, H. M. (2013). A parallel downloading study of local area network at universiti utara malaysia.

In Proceedings of the 4th International Conference on Computing and Informatics, ICOCI No. 106.

[Jin et al., 2012] Bo, J. (2012, March). Improved Parallel Downloading Algorithm for Large File Distribution. In Proceeding 2012 International Conference on Computer Distributed Control and Intelligent Environmental Monitoring (CDCIEM), PP. 373-376.

[Koo, Rosenberg and Dongyan, 2003] Koo, S. G., Rosenberg, C., & Xu, D. (2003, May). Analysis of parallel downloading for large file distribution. In *Distributed Computing Systems, 2003. FTDCS 2003. Proceedings. The Ninth IEEE Workshop on Future Trends of* (pp. 128-135). IEEE.

[Haitao, Zhenghu and Zunguo, 2005] Chen, H., Gong, Z., & Huang, Z. (2005, December). Parallel downloading algorithm for large-volume file distribution. In *Parallel and Distributed Computing, Applications and Technologies, 2005. PDCAT 2005. Sixth International Conference on* (pp. 745-749). IEEE.

[Karunadasa and Ranasinghe, 2009] Karunadasa, N. P., & Ranasinghe, D. N. (2009, December). Accelerating high performance applications with CUDA and MPI. In *Industrial and Information Systems (ICIIS), 2009 International Conference on* (pp. 331-336). IEEE.

[Online] Top500, "Japan's K Computer tops 10 pet flop/s to stay atop TOP500 list,"*http://top500.org/lists/2011/11*, Feb 2013.

[Baker, Buyyaand and Laforenza, 2002] Baker, M., Buyya, R., & Laforenza, D. (2002). Grids and Grid technologies for wide-area distributed computing. *Software: Practice and Experience*, *32*(15), 1437-1466.

[Venugopal, Buyya and Winton, 2006] Venugopal, S., Buyya, R., & Winton, L. (2006). A Grid service broker for scheduling e-Science applications on global data Grids. *Concurrency and Computation: Practice and Experience*, *18*(6), 685-699.

[Holmes and Kureshi, 2015] Holmes, V., & Kureshi, I. (2015). Developing High Performance Computing Resources for Teaching Cluster and Grid Computing courses. *Procedia Computer Science*, *51*, 1714-1723.

[Deelman, Singh, Livny, Berriman, 2008] Deelman, E., Singh, G., Livny, M., Berriman, B., & Good, J. (2008, November). The cost of doing science on the cloud: the montage example. In*Proceedings of the 2008 ACM/IEEE conference on Supercomputing* (p. 50). IEEE Press.

[Buyya, Yeoa, Venugopala, Broberga and Brandic, 2009] Buyya, R., Yeo, C. S., Venugopal, S., Broberg, J., & Brandic, I. (2009). Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation computer systems*,*25*(6), 599-616.

[Tchana, Palma, Etchevers and Hagimont, 2013] Tchana, A., De Palma, N., Etchevers, X., & Hagimont, D. (2013, January). Configuration challenges when migrating applications to a cloud: the JEE use case. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)* (p. 203). The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp).

[Mishra and Mishra, 2008] Mishra, D., & Mishra, A. (2008). Design issues in client-server software maintenance. *ACM SIGSOFT Software Engineering Notes*, *33*(5), 6.

[Hellerstein, Stonebraker and Hamilton, 2007] J Hellerstein, J. M., Stonebraker, M., & Hamilton, J. (2007). *Architecture of a database system*. Now Publishers Inc.

[Gopal, Nataraj, Ramamurthy and Sankaranarayan, 1996] Gopal, T. V., Nataraj, N. K., Ramamurthy, C., & Sankaranarayanan, V. (1996). Load balancing in heterogenous distributed systems. *Microelectronics Reliability*, *36*(9), 1279-1286.

[Galindo, Almeida, Manuel and Contelles, 2008] Galindo, I., Almeida, F., & Badía-Contelles, J. M. (2008). Dynamic load balancing on dedicated heterogeneous systems. In *Recent Advances in Parallel Virtual Machine and Message Passing Interface* (pp. 64-74). Springer Berlin Heidelberg.

[Haroon and Hussain, 2013] Haroon, M., & Husain, M. (2013). Analysis of a Dynamic Load Balancing in Multiprocessor System. *International Journal of Computer Science engineering and Information Technology Research*, *3*(1).

[Grama, Gupta, karypis and Kumar, 1994] Grama, A., Gupta, A., & Karypis, G. (1994). *Introduction to parallel computing: design and analysis of algorithms* (Vol. 400). Redwood City, CA: Benjamin/Cummings.

[Jordan and Alaghband, 2002] Jordan, L. E., & Alaghband, G. (2002). *Fundamentals of parallel processing*. Prentice Hall Professional Technical Reference.

[Sorin, Lemon, Eager and Vernon, 2003] Sorin, D. J., Lemon, J. L., Eager, D. L., & Vernon, M. K. (2003). Analytic evaluation of shared-memory architectures. *Parallel and Distributed Systems, IEEE Transactions on*, *14*(2), 166-180.

[Kandemir and Choudhary, 2002] Kandemir, M., Ramanujam, J., & Choudhary, A. (2002, June). Exploiting shared scratch pad memory space in embedded multiprocessor systems. In*Proceedings of the 39th annual Design Automation Conference* (pp. 219-224). ACM.

[Shivakumar and Jouppi, 2001] Shivakumar, P., & Jouppi, N. P. (2001). *Cacti 3.0: An integrated cache timing, power, and area model*. Technical Report 2001/2, Compaq Computer Corporation.

[Anderson, Kim and Herman, 2003] Anderson, J. H., Kim, Y. J., & Herman, T. (2003). Shared-memory mutual exclusion: major research trends since 1986. *Distributed Computing*, *16*(2-3), 75-110.

[Joseph, Pete and Alistair, 2006] Antony, J., Janes, P. P., & Rendell, A. P. (2006). Exploring thread and memory placement on NUMA architectures: Solaris and Linux, UltraSPARC/FirePlane and Opteron/HyperTransport. In *High Performance Computing-HiPC 2006* (pp. 338-352). Springer Berlin Heidelberg.

[Tanveer, Iqbal and Azam, 2011] Tanveer, M., Iqbal, M. A., & Azam, F. (2011). Using Symmetric Multiprocessor Architectures for High Performance Computing Environments.*International Journal of Computer Applications*, *27*(9).

[Kwak and Jhon, 2007] Kwak, J. W., & Jhon, C. S. (2007). Torus Ring: improving performance of interconnection network by modifying hierarchical ring. *Parallel Computing*,*33*(1), 2-20.

[Tripathy and Tripathy, 2011] Tripathy, M., & Tripathy, C. R. (2011). On a Virtual Shared Memory Cluster System with VirtualMachines. *International Journal of Computer and Electrical Engineering*, *3*(6), 754.

[Jung, Lim, Lee and Solihin, 2006] Jung, C., Lim, D., Lee, J., & Solihin, Y. (2006, April). Helper thread prefetching for loosely-coupled multiprocessor systems. In *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*(pp. 10-pp). Ieee.

[Chhabra and Singh, 2009] Lee, J., Jung, C., Lim, D., & Solihin, Y. (2009). Prefetching with helper threads for loosely coupled multiprocessor systems. *Parallel and Distributed Systems, IEEE Transactions on*, *20*(9), 1309-1324.

[Parhami, 2000] Parhami, B., & Kwai, D. M. (2000, June). Challenges in interconnection network design in the era of multiprocessor and massively parallel microchips. In *Proc. Int'l Conf. Communications in Computing* (pp. 241-246).

[Kim and Veidenbaum, 1999] Kim, S., & Veidenbaum, A. V. (1999). Interconnection network organization and its impact on performance and cost in shared memory multiprocessors.*Parallel Computing*, *25*(3), 283-309.

[Cheng and Chuang, 1994] Varietal hypercube: A new interconnection network topology for large scale multicomputer.

[Shi and Srimani, 2005] Shi, W., & Srimani, P. K. (2005). Hierarchical star: a new two level interconnection network. *Journal of Systems Architecture*, *51*(1), 1-14.

[Saad and Schultz, 1988] Saad, Y., & Schultz, M. H. (1988). Topological properties of hypercubes.*Computers, IEEE Transactions on*, *37*(7), 867-872.

[Amway and Latifi, 1991] El-Amawy, A., & Latifi, S. (1991). Properties and performance of folded hypercubes. *Parallel and Distributed Systems, IEEE Transactions on*, *2*(1), 31-42.

[Kumar and Patnaik, 1992] Kumar, J. M., & Patnaik, L. M. (1992). Extended hypercube: A hierarchical interconnection network of hypercubes. *Parallel and Distributed Systems, IEEE Transactions on*, *3*(1), 45-57.

[Efe et al., 1991] Efe, K. (1991). A variation on the hypercube with lower diameter. *Computers, IEEE Transactions on*, *40*(11), 1312-1316.

[Loh, Hsu and Pan, 2005] Efe, K. (1991). A variation on the hypercube with lower diameter. *Computers, IEEE Transactions on*, *40*(11), 1312-1316.

[Zhang et al., 2002] Zhang, Y. Q. (2002). Folded-crossed hypercube: a complete interconnection network. *Journal of systems architecture*, *47*(11), 917-922.

[Efe et al., 1992] Efe, K. (1992). The crossed cube architecture for parallel computation.*Parallel and Distributed Systems, IEEE Transactions on*, *3*(5), 513-524.

[Ghose and Desai, 1995] Ghose, K., & Desai, K. R. (1995). Hierarchical cubic networks. *Parallel and Distributed Systems, IEEE Transactions on*, *6*(4), 427-435.

[Kumar et al., 2012] Kumar, N. (2012). Simulation Study for Performance and Prediction of Parallel Computers. *BIJIT is indexed with the following publishers*, 500.

[Khan, Siddiqui and Samad, 2013] Khan, Z. A., Siddiqui, J., & Samad, A. (2013). Performance Analysis of Massively Parallel Architectures. *BIJITBVICAM's International Journal of Information Technology*, *5*(1), 563-568.

[Patel, Parandkar, Katiyal and Agarwal, 2011] Patel, S., Parandkar, P., Katiyal, S., & Agrawal, A. (2011). Exploring alternative topologies for network-on-chip architectures. *Int J Inf Technol*, *3*, 372-376.

[Patel, Kasalik and McCrosky, 2006] Patel, A., Kusalik, A., & McCrosky, C. (2000). Area-efficient VLSI layouts for binary hypercubes. *Computers, IEEE Transactions on*, *49*(2), 160-169.

[Li and Peng, 2000] Li, Y., & Peng, S. (2000, December). Dual-cubes: a new interconnection network for high-performance computer clusters. In *Proceedings of the 2000 international computer symposium, workshop on computer architecture* (pp. 51-57).

[Adhikari and Tripathy, 2008] Adhikari, N., & Tripathy, C. R. (2008, December). Folded dualcube: A new interconnection topology for parallel systems. In *Information Technology, 2008. ICIT'08. International Conference on* (pp. 75-78). IEEE.

[Peng and Chu, 2002] Li, Y., Peng, S., & Chu, W. (2002). Metacube: a new interconnection network for large scale parallel systems. *Australian Computer Science Communications*, *24*(3), 29-36.

[Adhikari and Tripathy, 2009] Adhikari, N., & Tripathy, C. R. (2009, March). Folded Metacube: An Efficient Large Scale Parallel Interconnection Network. In *Advance Computing Conference, 2009. IACC 2009. IEEE International* (pp. 1281-1285). IEEE.

[Adhikari and Tripathy, 2010] Adhikari, N., & Tripathy, C. R. (2010). The folded crossed cube: a new interconnection network for parallel systems. *International Journal of Computer Applications (0975–8887)*, *4*(3), 43-50.

[Adhikari and Tripathy, 2014] Adhikari, N., & Tripathy, C. R. (2014). Star-crossed cube: an alternative to star graph. *Turkish Journal of Electrical Engineering & Computer Sciences*,*22*(3), 719-734.

[Monemizadeh and Azad, 2005] Monemizadeh, M., & Sarbazi-Azad, H. (2005, March). The necklace-hypercube: a well scalable hypercube-based interconnection network for multiprocessors. In *Proceedings of the 2005 ACM symposium on Applied computing* (pp. 729-733). ACM.

[Rafiq, Kumar and Gupta, 1999] Rafiq, M. Q., Kumar, P., & Gupta, J. P. (1995). A Novel Tree-Structured Multiprocessor Network. In *Proceedings of International Conference of on Robotics Vision and Parallel Processing for Automation, Malaysia* (Vol. 2, pp. 576-585).

[Mohanty, Ray, Patro and Tripathy, 2008] Mohanty, S. P., Ray, B. B., Patro, S. N., & Tripathy, A. R. (2008, December). Topological properties of a new fault tolerant interconnection network for parallel computer. In *International Conference on Information Technology* (pp. 36-40). IEEE.

[Youyao, Jungang and Huimin, 2008] Liu, Y., Han, J., & Du, H. (2008). A hypercube-based scalable interconnection network for massively parallel computing. *Journal of Computers*, *3*(10), 58-65.

[Adhikari and Tripathy, 2009] Adhikari, N., & Tripathy, C. R. (2009, August). Extended crossed cube: an improved fault tolerant interconnection network. In *2009 Fifth International Joint Conference on INC, IMS and IDC* (pp. 86-91). IEEE.

[Akers, Harel and Krishnamurthy, 1987] Akers, S. B., Harel, D., & Krishnamurthy, B. (1994, June). The star graph: An attractive alternative to the n-cube. In *Interconnection networks for high-performance parallel computers* (pp. 145-152). IEEE Computer Society Press.

[Liu, Fang and Wu, 2004] Liu, Y. C., Fang, J. F., & Wu, C. C. (2004). On Extensibilities Of Interconnection Networks. *IE.(I) Journal-CP*, 13-16.

[Dhakal, Paskaleva, Hayat, Schamiloglu and Abdallah, 2003] Dhakal, S., Paskaleva, B. S., Hayat, M. M., Schamiloglu, E., & Abdallah, C. T. (2003, December). Dynamical discrete-time load balancing in distributed systems in the presence of time delays. In *Decision and Control, 2003. Proceedings. 42nd IEEE Conference on* (Vol. 5, pp. 5128-5134). IEEE

[Chen, Liao, Hsie and Liao, 2008] Chen, J. C., Liao, G. X., Hsie, J. S., & Liao, C. H. (2008). A study of the contribution made by evolutionary learning on dynamic load-balancing problems in distributed computing systems. *Expert Systems with Applications*, *34*(1), 357-365.`

[Altman, Ayesha and Prabhu, 2001] Altman, E., Ayesta, U., & Prabhu, B. J. (2011). Load balancing in processor sharing systems. *Telecommunication Systems*, *47*(1-2), 35-48.

[Sharma and Gahlawat, 2013] Gahlawat, M., & Sharma, P. (2013). Analysis and Performance Assessment of CPU Scheduling Algorithms in Cloud using Cloud Sim. *Analysis*, *5*(9).

[Bansal, Kothari and Hota, 2011] Bansal, S., Kothari, B., & Hota, C. (2011). Dynamic task-scheduling in grid computing using prioritized round robin algorithm. *International Journal of Computer Science Issues(IJCSI)*, *8*(2).

[Kaur and Kaur, 2013] Kaur, R., & Kaur, R. (2013). Multiprocessor scheduling using task duplication based scheduling algorithms: A review paper. *International Journal of Application or Innovation in Engineering and Management*, *2*(4), 311-317.

[Samad, Rafiq and Farooq, 2012] Samad, A., Rafiq, M. Q., & Farooq, O. (2012, December). Multi-stage scheduling scheme for massively parallel systems. In *Software Engineering and Mobile Application Modelling and Development (ICSEMA 2012), International Conference on* (pp. 1-6). IET.

[Bertogna, Cirinei and Lipari, 2009] Bertogna, M., Cirinei, M., & Lipari, G. (2009). Schedulability analysis of global scheduling algorithms on multiprocessor platforms. *Parallel and Distributed Systems, IEEE Transactions on*, *20*(4), 553-566.

[Sharma, Singh and Sharma, 2008] Sharma, S., Singh, S., & Sharma, M. (2008). Performance analysis of load balancing algorithms. *World Academy of Science, Engineering and Technology*, *38*, 269-272.

[Zaki, Li and Parthasarathy, 1996] Zaki, M. J., Li, W., & Parthasarathy, S. (1996, August). Customized dynamic load balancing for a network of workstations. In *High Performance Distributed Computing, 1996., Proceedings of 5th IEEE International Symposium on* (pp. 282-291). IEEE.

[Andrade, DeMello, Dodonov, Senger, Yang and Li, 2008] De Mello, R. F., Dodonov, E., Senger, L. J., Yang, L. T., & Li, K. C. (2008, July). Toward an efficient middleware for multithreaded applications in computational grid. In *Computational Science and Engineering, 2008. CSE'08. 11th IEEE International Conference on* (pp. 147-154). IEEE.

[Barbosa and Moreira, 2011] J.G. Barbosa and B. Moreira, "Dynamic Task Scheduling Algorithm With load Balancing For Heterogeneous Computing System,"*Parallel computing*, pp. 428-438, 2011.

[Fang, Wang and Ge, 2010] Fang, Y., Wang, F., & Ge, J. (2010). A task scheduling algorithm based on load balancing in cloud computing. In *Web Information Systems and Mining*(pp. 271-277). Springer Berlin Heidelberg.

[Braun et al., 2001] Braun, T. D., Siegel, H. J., Beck, N., Bölöni, L. L., Maheswaran, M., Reuther, A. I.,. & Freund, R. F. (2001). A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. *Journal of Parallel and Distributed computing*, *61*(6), 810-837.

[Roy et al., 2012] Roy, P., Alam, M., Ul, M., & Das, N. (2012). Heuristic based task scheduling in multiprocessor systems with genetic algorithm by choosing the eligible processor. *arXiv preprint arXiv:1208.1922*.

[Yang, Zhou, Sun and Cruickshank, 2013] Y Yang, Y., Zhou, Y., Sun, Z., & Cruickshank, H. (2013). Heuristic scheduling algorithms for allocation of virtualized network and computing resources.

[Waldspurger and Weihl, 1995] Waldspurger, C. A., & Weihl, W. E. (1995). *Stride scheduling: Deterministic proportional share resource management*. Massachusetts Institute of Technology. Laboratory for Computer Science.

[Dhodhi, Ahmad, Yatama and Ahmad, 2002] Dhodhi, M. K., Ahmad, I., Yatama, A., & Ahmad, I. (2002). An integrated technique for task matching and scheduling onto distributed heterogeneous computing systems. *Journal of parallel and distributed computing*, *62*(9), 1338-1361.

[Ghafoor and Yang, 1993] Ghafoor, A., & Yang, J. (1992). Distributed heterogeneous supercomputing management system. *ECE Technical Reports*, 270.

[Zhang, Jiang and Li, 2009] Zhang, D., Jiang, C., & Li, S. (2009). A fast adaptive load balancing method for parallel particle-based simulations. *Simulation Modelling Practice and Theory*, *17*(6), 1032-1042.

[Dobber, Mei and Koole, 2009] Dobber, M., Van Der Mei, R., & Koole, G. (2009). Dynamic load balancing and job replication in a global-scale grid environment: A comparison. *Parallel and Distributed Systems, IEEE Transactions on*, *20*(2), 207-218.

[Colajanni and Dias, 1998] Colajanni, M., Yu, P. S., & Dias, D. M. (1998). Analysis of task assignment policies in scalable distributed Web-server systems. *Parallel and Distributed Systems, IEEE Transactions on*, *9*(6), 585-600.

[Kwokand Ahmad, 1999] Kwok, Y. K., & Ahmad, I. (1999). Static scheduling algorithms for allocating directed task graphs to multiprocessors. *ACM Computing Surveys (CSUR)*,*31*(4), 406-471.

[Balter, Crovella and Murta, 1998] Harchol-Balter, M., Crovella, M. E., & Murta, C. D. (1999). On choosing a task assignment policy for a distributed server system. *Journal of Parallel and Distributed Computing*, *59*(2), 204-228.

[Bahnasawy, Omara, Koutb and Mosa, 2011] Bahnasawy, N. A., Omara, F., Koutb, M. A., & Mosa, M. (2011). A new algorithm for static task scheduling for heterogeneous distributed computing systems. *African Journal of Mathematics and Computer Science Research*,*4*(6), 221-234.

[Amalarethinamand and Josphin, 2015] Amalarethinam, D. G., & Josphin, A. M. (2015). Dynamic Task Scheduling Methods in Heterogeneous Systems: A Survey. *International Journal of Computer Applications*, *110*(6).

[Stigge, Ekberg, Guan and Yi, 2011] Stigge, M., Ekberg, P., Guan, N., & Yi, W. (2011, April). The digraph real-time task model. In *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2011 17th IEEE* (pp. 71-80). IEEE.

[Kafil and Ahmad, 1998] Kafil, M., & Ahmad, I. (1998). Optimal task assignment in heterogeneous distributed computing systems. *Concurrency, IEEE*, *6*(3), 42-50.

[Amalarethinam and Mary, 2011] Amalarethinam, D. G., & Mary, G. J. (2011). A new DAG based Dynamic Task Scheduling Algorithm (DYTAS) for Multiprocessor Systems. *International Journal of Computer Applications*, *19*(8), 24-28.

[Abdelkader and Omara, 2012] Abdelkader, D. M., & Omara, F. (2012). Dynamic task scheduling algorithm with load balancing for heterogeneous computing system. *Egyptian Informatics Journal*, *13*(2), 135-145.

[Zheng, Meneses, Bhatele and Kale, 2010] Zheng, G., Meneses, E., Bhatele, A., & Kale, L. V. (2010, September). Hierarchical load balancing for Charm++ applications on large supercomputers. In *Parallel Processing Workshops (ICPPW), 2010 39th International Conference on* (pp. 436-444). IEEE.

[Grosu and Chronopoulos, 2004] Grosu, D., & Chronopoulos, A. T. (2004). Algorithmic mechanism design for load balancing in distributed systems. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, *34*(1), 77-84.

[Sahoo, Kumar and Jena, 2013] Sahoo, B., Kumar, D., & Jena, S. K. (2013). Analysing the impact of heterogeneity with greedy resource allocation algorithms for dynamic load balancing in heterogeneous distributed computing system. *International Journal of Computer Applications*, *62*(19).

[Cosenza, Cordasco, Chiara and Scarano, 2011] Cosenza, B., Cordasco, G., De Chiara, R., & Scarano, V. (2011, February). Distributed load balancing for parallel agent-based simulations. In *Parallel, Distributed and Network-Based Processing (PDP), 2011 19th Euromicro International Conference on* (pp. 62-69). IEEE.

[Spies et al., 1996] Spies, F. (1996). Modeling of optimal load balancing strategy using queueing theory. *Microprocessing and microprogramming*, *41*(8), 555-570.

[Li and Kameda, 1998] Li, J., & Kameda, H. (1998). Load balancing problems for multiclass jobs in distributed/parallel computer systems. *Computers, IEEE Transactions on*,*47*(3), 322-332.

[Ali, Siegel and Hensgen, 2000] Ali, S., Siegel, H. J., Maheswaran, M., & Hensgen, D. (2000). Task execution time modeling for heterogeneous computing systems. In *Heterogeneous Computing Workshop, 2000.(HCW 2000) Proceedings. 9th* (pp. 185-199). IEEE.

[Daouas, Ghedira and Muller, 1995] Daonas, T., Ghedira, K., & Muller, J. P. Distributed Flow Shop Scheduling Problem Global versus Local Optimization.

[Mujumdar, 1992] Majumdar, S. (1992, April). The performance of local and global scheduling strategies in multiprogrammed parallel systems. In *Computers and Communications, 1992. Conference Proceedings., Eleventh Annual International Phoenix Conference on* (pp. 55-62). IEEE.

[Karatza and Hilzer, 2002] Zhou, S. (1992). Lsf: Load sharing in large heterogeneous distributed systems. In *I Workshop on Cluster Computing*.

[Attiya and Hamam, 2006] Attiya, G., & Hamam, Y. (2006). Task allocation for maximizing reliability of distributed systems: a simulated annealing approach. *Journal of parallel and Distributed Computing*, *66*(10), 1259-1266.

[Sahoo, Jain, Iyer and Dill, 2005] Sahoo, D., Jain, J., Iyer, S., & Dill, D. (2005). A new reachability algorithm for symmetric multi-processor architecture. In *Automated Technology for Verification and Analysis* (pp. 26-38). Springer Berlin Heidelberg.

[Wang, Zheng and Xiong, 1997] Wang, D., Zheng, W., & Xiong, J. (1997, December). Research on cluster of workstations. In *Parallel Architectures, Algorithms, and Networks, 1997.(I-SPAN'97) Proceedings., Third International Symposium on* (pp. 275-281). IEEE.

[Simon et al., 2009] Dongarra, J. (2008). Future directions in high performance computing. *Talk presented at Google, Mountain View, CA*.

[Nguyen and Desideri, 2012] Nguyen, T., & Desideri, J. A. (2012). A Distributed Workflow Platform for High-Performance Simulation. *International Journal On Advances in Intelligent Systems*, *4*(3&4), 82-101.

[Dobber, Kooleand and Mei, 2004] Dobber, M., Koole, G., & Van Der Mei, R. (2004). Dynamic load balancing for a grid application. In *High Performance Computing-HiPC 2004* (pp. 342-352). Springer Berlin Heidelberg.

[Banicescu and Velusamy, 2002] Banicescu, I., & Velusamy, V. (2002, April). Load balancing highly irregular computations with the adaptive factoring. In *Parallel and Distributed Processing Symposium., Proceedings International, IPDPS 2002, Abstracts and CD-ROM* (pp. 12-pp). IEEE..

[Attiya et al., 2004] Attiya, G., & Hamam, Y. (2004, February). Two phase algorithm for load balancing in heterogeneous distributed systems. In *Parallel, Distributed and Network-Based Processing, 2004. Proceedings. 12th Euromicro Conference on* (pp. 434-439). IEEE.

[Beaumont, Carter, Ferrante, Legrand, Marchal and Robert, 2008] Beaumont, O., Carter, L., Ferrante, J., Legrand, A., Marchal, L., & Robert, Y. (2008). Centralized versus distributed schedulers for bag-of-tasks applications. *Parallel and Distributed Systems, IEEE Transactions on*, *19*(5), 698-709.

[Chandra and Shenoy, 2008] Chandra, A., & Shenoy, P. (2008). Hierarchical scheduling for symmetric multiprocessors. *Parallel and Distributed Systems, IEEE Transactions on*,*19*(3), 418-431.

[Lin and Raghavendra, 1992] Lin, H. C., & Raghavendra, C. S. (1992). A dynamic load-balancing policy with a central job dispatcher (LBC). *Software Engineering, IEEE Transactions on*, *18*(2), 148-158.

[Ishfaqand and Ghafoor, 1991] Ahmad, I., & Ghafoor, A. (1991). Semi-distributed load balancing for massively parallel multicomputer systems. *Software Engineering, IEEE Transactions on*, *1*.

[Subrata, Zomaya, and landfeldet, 2008] Subrata, R., Zomaya, A. Y., & Landfeldt, B. (2008). A cooperative game framework for QoS guided job allocation schemes in grids. *Computers, IEEE Transactions on*, *57*(10), 1413-1422.

[Bahi, Couturier and Vernie, 2005] Bahi, J., Couturier, R., & Vernier, F. (2005). Synchronous distributed load balancing on dynamic networks. *Journal of Parallel and Distributed Computing*, *65*(11), 1397-1405.

[Vinay, Abhijit, Saifulla, Jayashree and Anitha, 2011] Vinay, A., Abhijit, K. G., Saifulla, M., Jayashree, D., & Anitha, T. N. (2011, February). A comparative analysis of centralized and distributed dynamic load balancing algorithms for cluster based video-on-demand systems. In*Proceedings of the International Conference & Workshop on Emerging Trends in Technology* (pp. 351-356). ACM.

[Azzoni and Down, 2009] Al-Azzoni, I., & Down, D. G. (2009, November). Decentralized load balancing for heterogeneous grids. In *Future Computing, Service Computation, Cognitive, Adaptive, Content, Patterns, 2009. COMPUTATIONWORLD'09. Computation World:* (pp. 545-550). IEEE.

[Jain and Gupta, 2009] Jain, P., & Gupta, D. (2009). An algorithm for dynamic load balancing in distributed systems with multiple supporting nodes by exploiting the interrupt service. *International Journal of Recent Trends in Engineering*, *1*(1), 232-236.

[Kremin and Kramer, 2005] Kremien, O., & Kramer, J. (1992). Methodical analysis of adaptive load sharing algorithms. *Parallel and Distributed Systems, IEEE Transactions on*,*3*(6), 747-760.

[Zomaya and Yee, 2001] Zomaya, A. Y., & Teh, Y. H. (2001). Observations on using genetic algorithms for dynamic load-balancing. *Parallel and Distributed Systems, IEEE Transactions on*, *12*(9), 899-911.

[Mitchell et al., 2004] Mitchell, M. (2004). *An introduction to genetic algorithms*. MIT press.

[Jingyi et al., 2010] Ma. Jingyi, "Ma, J. (2010, August). A Novel Heuristic Genetic Load Balancing Algorithm in Grid Computing. In *Intelligent Human-Machine Systems and Cybernetics (IHMSC), 2010 2nd International Conference on* (Vol. 2, pp. 166-169). IEEE.

[Rafiq, Padamand, Gupta, 1999] Rafiq, M. Q., Kumar, P., & Gupta, J. P. (1999). A Novel Tree-Structured Multiprocessor Network. In *Proceedings of International Conference of on Robotics Vision and Parallel Processing for Automation, Malaysia* (Vol. 2, pp. 576-585)..

[Ravikanth, Sastryand and Ramakrishanan, 1988] Ravikanth, K., Sastry, P. S., Ramakrishnan, K. R., & Venkatesh, Y. V. (1988). A reduction architecture for the optimal scheduling of binary trees.*Future Generation Computer Systems*, *4*(3), 225-233..

[Reddy et al., 1993] V. A. Reddy,"Design of parallel reduction model for the implementation of functional languages,"*Ph.D Thesis*, IIT, Roorkee, 1993.

[LeMairand and Reeves, 1993] Willebeek-LeMair, M. H., & Reeves, A. P. (1993). Strategies for dynamic load balancing on highly parallel computers. *Parallel and Distributed Systems, IEEE Transactions on*, *4*(9), 979-993.

[Rafiq et al., 1995] Rafiq, M. Q. (1995). *Studies on the Performance Evaluation of a Linearly Extensible Multiprocessor Network* (Doctoral dissertation, Ph. D thesis, Univ. of Roorkee).

[Bari and Meshram, 2013] P. H. Bari and B. B. Meshram, "Load Balancing In Distributed Computing," *Journal of Engineering, Computers & Applied Sciences,* Vol. 2, No.6, PP. 43-51,2013.

[Manaullah et al., 2013] Manaullah,"A Δ-Based Linearly Extensible Multiprocessor Network," *International Journal of Computer Science and Information Technologies,* Vol. 4, No. 5, PP. 700-707, 2013.

[Bhatti, Belleudy and Auguin, 2011] Bhatti, M. K., Belleudy, C., & Auguin, M. (2011). Two-level Hierarchical Scheduling Algorithm for Real-time Multiprocessor Systems. *Journal of Software*, *6*(11), 2308-2320.

[I. Shin, A. Easwaran and I. Lee] Shin, I., Easwaran, A., & Lee, I. (2008, July). Hierarchical scheduling framework for virtual clustering of multiprocessors. In *Real-Time Systems, 2008. ECRTS'08. Euromicro Conference on* (pp. 181-190). IEEE.

[Lipari and Bini, 2005] Lipari, G., & Bini, E. (2005). A methodology for designing hierarchical scheduling systems. *Journal of Embedded Computing*, *1*(2), 257-269.

[Lin and Keller, 1987] Lin, F. C., & Keller, R. M. (1987). The gradient model load balancing method.*Software Engineering, IEEE Transactions on*, (1), 32-38.

[Bronevich and Meyer, 2008] Bronevich, A. G., & Meyer, W. (2008). Load balancing algorithms based on gradient methods and their analysis through algebraic graph theory. *Journal of Parallel and Distributed Computing*, *68*(2), 209-220.

[Bojańczyk and Colcombet, 2006, 2006] M. Bojańczyk and T. Colcombet, "Tree-walking automata cannot be feminised,"*Theoretical Computer Science*, Vol. 350, No. (2-3), PP. 164–173, 2006.

[Samad and Rafiq, 2005] A. Samad and M. Q. Rafiq, "A Novel Server Architecture for Networking," *In proceedings of International Conference on robotics, vision, information and signal processing (ROVISP2005),* University Sains, Malaysia, PP. 1029-1032, 2005.

[Wanga, Zhangb, Sua, Wanga, Chena, Jia and Shia, 2014] Wang, Y., Zhang, Y., Su, Y., Wang, X., Chen, X., Ji, W., & Shi, F. (2014). An adaptive and hierarchical task scheduling scheme for multi-core clusters.*Parallel Computing*, *40*(10), 611-627.

[Tharani and Deepa, 2012] THARANI, R., & DEEPA, K. ADAPTIVE GRID JOB SCHEDULING WITH IMPROVING HIERARCHICAL LOAD BALANCED ALGORITHM.

[Page and Naughton, 2005] Page, A. J., & Naughton, T. J. (2005). Framework for task scheduling in heterogeneous distributed computing using genetic algorithms. *Artificial Intelligence Review*, *24*(3-4), 415-429.

[Zomaya, Lee and Olariu, 2001] Zomaya, A. Y., Lee, R. C., & Olariu, S. T. E. P. H. A. N. (2001). An introduction to genetic-based scheduling in parallel processor systems.*Solutions to Parallel and Distributed Computing Problems*, 111-133.

[Omara and Arafa, 2010] Omara, F. A., & Arafa, M. M. (2010). Genetic algorithms for task scheduling problem. *Journal of Parallel and Distributed Computing*, *70*(1), 13-22.

[Guzek, Pecero, Dorronsoro and Bouvry, 2014] M. Guzek, J. E. Pecero, B. Dorronsoro and P. Bouvry, "Multi-objective evolutionary algorithms for energy-aware scheduling on distributed computing systems," *Applied Soft Computing*, Vol. 24, PP. 432–446, 2014.

[Rajak, Katti and Rajak, 2013] Guzek, M., Pecero, J. E., Dorronsoro, B., & Bouvry, P. (2014). Multi-objective evolutionary algorithms for energy-aware scheduling on distributed computing systems. *Applied Soft Computing*, *24*, 432-446..

[Kwok and Ahmad, 1996] Kwok, Y. K., & Ahmad, L. (1996). Dynamic critical-path scheduling: An effective technique for allocating task graphs to multiprocessors. *Parallel and Distributed Systems, IEEE Transactions on*, *7*(5), 506-521.

[Mahafzah , Jaradat, 2010] Mahafzah, B. A., & Jaradat, B. A. (2010). The hybrid dynamic parallel scheduling algorithm for load balancing on Chained-Cubic Tree interconnection networks. *The Journal of Supercomputing*, *52*(3), 224-252.

[Rosemarry, Singh, Singhal and Sisodia, 2012] Rosemarry, P., Singh, R., Singhal, P., & Sisodia, D. (2012). Grouping Based Job Scheduling Algorithm Using Priority Queue And Hybrid Algorithm In Grid Computing. *International Journal of Grid Computing & Applications*, *3*(4), 55.

[Online] Depth-Limited Search, *https://en.wikipedia.org/wiki/Depth-limited_search.*

[Esfahanianand, Sagan, 1993] N. M. L. Esfahanianand E. B. Sagan, "The Twisted N-Cube with application to multiprocessing,"*IEEE Transaction on Computer*, Vol. 40, No. 4, PP. 88-93, 1993.

[Rajput and Kumari, 2012] Rajput, P., & Kumari, V. (2012). Modelling and Evaluation of Multiprocessor Architecture. *International Journal of Computer Applications*, *51*(21).

[Rashed et al., 2012] Rashed, A. N. Z. (2012). Very Large Scale Optical Interconnect Systems For Different Types of Optical Interconnection Networks. *International Journal of Computer Network and Information Security*, *4*(3), 62.

[Awwad, Ayyoub and Khaoua, 2003] A Awwad, A. M., Al-Ayyoub, A., & Ould-Khaoua, M. (2003). On the topological properties of the arrangement–star network. *Journal of Systems Architecture*, *48*(11), 325-336.

[Youyao, Jungang and Huimin, 2008] Liu, Y., Han, J., & Du, H. (2008). A hypercube-based scalable interconnection network for massively parallel computing. *Journal of Computers*, *3*(10), 58-65.

[Tripathy et al., 2004] Tripathy, C. R. (2004). Star-cube: A new fault-tolerant interconnection topology for massively parallel systems. *JOURNAL-INSTITUTION OF ENGINEERS INDIA PART ET ELECTRONICS AND TELECOMMUNICATIONS ENGINEERING DIVISION*, 83-92.

[Khan, Siddiqui and Samad, 2013] Khan, Z. A., Siddiqui, J., & Samad, A. (2013). Topological Evaluation Of Variants Hypercube Network. *Asian Journal of Computer Science and Information Technology*, *3*(9), 125-128.

[Peter, Hsu and Pan, 2011] Loh, P. K., Hsu, W. J., & Pan, Y. (2005). The exchanged hypercube. *IEEE Transactions on Parallel & Distributed Systems*, (9), 866-874.

[Adhikari and Tripathy, 2011] Tripathy, C. R., & Adhikari, N. (2011). On a new multicomputer interconnection topology for massively parallel systems. *arXiv preprint arXiv:1108.1462*.

[Akers and Krishnamurthy, 1987] Akers, S. B., & Krishnamurthy, B. (1987, May). The fault tolerance of star graphs. In *Proc. 2nd International Conference on Supercomputing* (pp. 270-276).

[Samad, Rafiq and Farooq, 2010] Samad, A., Rafiq, M. Q., & Farooq, O. (2010, January). LEC: An Efficient Scalable Parallel Interconnection Network. In *proceeding International Conference on Emerging Trends in Computer Science, Communication and Information Technology* (pp. 453-458).

[Ayyouband and Day, 1998] Al-Ayyoub, A. E., & Day, K. (1998). The hyperstar interconnection network.*Journal of Parallel and Distributed Computing*, *48*(2), 175-199.

[Abuelrub et al., 2008] Abuelrub, E. (2008). A Comparative Study on the Topological Properties of the Hyper-Mesh Interconnection Network.

[Cardellini, Casalicchio, Colajanni and YU, 2002] Cardellini, V., Casalicchio, E., Colajanni, M., & Yu, P. S. (2002). The state of the art in locally distributed Web-server systems. *ACM Computing Surveys (CSUR)*, *34*(2), 263-311.

[Khan, Siddiqui and Samad, 2015] Khan, Z. A., Siddiqui, J., & Samad, A. (2015). Linear Crossed Cube (LCQ): A New Interconnection Network Topology for Massively Parallel System.*International Journal of Computer Network and Information Security (IJCNIS)*, *7*(3), 18.

[Khan, Siddiqui and Samad, 2014] Khan, Z. A., Siddiqui, J., & Samad, A. (2014, December). A novel multiprocessor architecture for massively parallel system. In *Parallel, Distributed and Grid Computing (PDGC), 2014 International Conference on*(pp. 466-471). IEEE.

[Jia, Veeravalli and Weissman, 2010] Jia, J., Veeravalli, B., & Weissman, J. (2010). Scheduling multisource divisible loads on arbitrary networks. *Parallel and Distributed Systems, IEEE Transactions on*, *21*(4), 520-531.

[Teodorescu and Torrellas, 2008] Teodorescu, R., & Torrellas, J. (2008, June). Variation-aware application scheduling and power management for chip multiprocessors. In *ACM SIGARCH computer architecture news* (Vol. 36, No. 3, pp. 363-374). IEEE Computer Society.

[Martelli and Bonuccelli, 2013] Martelli, F., & Bonuccelli, M. A. (2013). Minimum Message Waiting Time Scheduling in Distributed Systems. *Parallel and Distributed Systems, IEEE Transactions on*, *24*(9), 1797-1806.

[Dodonov and Mello, 2010] Dodonov, E., & de Mello, R. F. (2010). A novel approach for distributed application scheduling based on prediction of communication events. *Future Generation Computer Systems*, *26*(5), 740-752.

[Lan, E. V. Taylor and Bryan, 2002] Lan, Z., Taylor, V. E., & Bryan, G. (2002). A novel dynamic load balancing scheme for parallel systems. *Journal of Parallel and Distributed Computing*,*62*(12), 1763-1781.

[Hwang, Gen and Katayama, 2008] Hwang, R., Gen, M., & Katayama, H. (2008). A comparison of multiprocessor task scheduling algorithms with communication costs.*Computers & Operations Research*, *35*(3), 976-993.

[Kang, He and Song, 2011] Kang, Q., He, H., & Song, H. (2011). Task assignment in heterogeneous computing systems using an effective iterated greedy algorithm. *Journal of Systems and Software*, *84*(6), 985-992.

[DeMello, Senger and Yang, 2006] de Mello, R. F., Senger, L. J., & Yang, L. T. (2006). Performance Evaluation of Route: A Load Balancing Algorithm for Grid Computing. *RITA*, *13*(1), 87-108.

[Ishii, Mello and Yang, 2007] R. P. Ishii, R. F. De Mello and L. T. Yang, "A Complex Network Based Approach for Job Scheduling in Grid Environments,"HPC inLecture Notes in Computer Science, Vol. 4782, Springer, pp. 204-2015, 2007.

[Jin, Jespersen, Mehrotra, Biswas, Huang and Chapman, 2011] Ishii, R. P., De Mello, R. F., & Yang, L. T. (2007). A complex network-based approach for job scheduling in grid environments. In *High Performance Computing and Communications* (pp. 204-215). Springer Berlin Heidelberg.

[Zoghdy, 2012] El-Zoghdy, S. F. (2012). A hierarchical load balancing policy for grid computing environment. *International Journal of Computer Network and Information Security*, *4*(5), 1.

[Guan, Yai, Deng, Gu and Yu, 2011] Guan, N., Yi, W., Deng, Q., Gu, Z., & Yu, G. (2011). Schedulability analysis for non-preemptive fixed-priority multiprocessor scheduling. *Journal of Systems Architecture*, *57*(5), 536-546.

[Canfora and Cerulo, 2014] G. Canfora and L.Cerulo, "A Taxonomy of Information Retrieval Information Models and Tools,"*Journal of Computing and Information Technology*, Vol. 12, No.3, PP. 175-194, 2004.

[Kharwar, Viyas and Shah, 2014] Vyas, T., Kharwar, C., & Shah, V. (2014). Content Based Parallel Information Retrieval for Text Files–Exploiting the Multiprocessor Functionality.

[Rao and Nagaraj, 2010] Rao, G. N., & Nagaraj, S. (2010). Client Level Framework for Parallel Downloading of Large File Systems. *International Journal of Computer Applications*, *3*(2).

[Zhu, Yu, Wang, Tan, Low, 2010] Zhu, Y., Yu, Y., Wang, W. Y., Tan, S. S., & Low, T. C. (2010, July). A balanced allocation strategy for file assignment in parallel i/o systems. In*Networking, Architecture and Storage (NAS), 2010 IEEE Fifth International Conference on* (pp. 257-266). IEEE.

[Cho, Jin, Lee and Schwan, 2014] Cho, J. Y., Jin, H. W., Lee, M., & Schwan, K. (2014). Dynamic core affinity for high-performance file upload on Hadoop Distributed File System. *Parallel Computing*, *40*(10), 722-737.

[Long, Zhao, Chen and Tang, 2015] Long, S., Zhao, Y., Chen, W., & Tang, Y. (2015). A prediction-based dynamic file assignment strategy for parallel file systems. *Parallel Computing*, *41*, 1-13.

[Zeitoun, Jamjoom and Gendy, 2002] Zeitoun, A., Jamjoom, H., & El-Gendy, M. (2002, February). Scalable parallel-access for mirrored servers. In *APPLIED INFORMATICS-PROCEEDINGS-* (No. 3, pp. 93-98). UNKNOWN.

[Song, Yin, Chen and Sun, 2011] Song, H., Yin, Y., Chen, Y., & Sun, X. H. (2011, June). A cost-intelligent application-specific data layout scheme for parallel file systems. In*Proceedings of the 20th international symposium on High performance distributed computing* (pp. 37-48). ACM.

[Zeng and Veeravalli, 2006] Zeng, Z., & Veeravalli, B. (2006). Design and performance evaluation of queue-and-rate-adjustment dynamic load balancing policies for distributed networks. *Computers, IEEE Transactions on*, *55*(11), 1410-1422.

[Rodriguez and Biersack, 2002] Rodriguez, P., & Biersack, E. W. (2002). Dynamic parallel access to replicated content in the Internet. *IEEE/ACM Transactions on Networking (TON)*, *10*(4), 455-465.

[shen, and Xu, 2209] Shen, H., & Xu, S. (2009). Coordinated en-route web caching in multiserver networks. *Computers, IEEE Transactions on*, *58*(5), 605-619.

[Foglia, Giorgi, and Prete, 2000] Foglia, P., Giorgi, R., & Prete, C. A. (2000, January). Performance analysis of electronic commerce multiprocessor server. In *System Sciences, 2000. Proceedings of the 33rd Annual Hawaii International Conference on* (pp. 9-pp). IEEE.

[Barroso, Dean and Hlzle, 2003] Barroso, L. A., Dean, J., & Hölzle, U. (2003). Web search for a planet: The Google cluster architecture. *Micro, Ieee*, *23*(2), 22-28.

[Ranjanand and Knightly, 2008] Ranjan, S., & Knightly, E. (2008). High-performance resource allocation and request redirection algorithms for web clusters. *Parallel and Distributed Systems, IEEE Transactions on*, *19*(9), 1186-1200.

[Liddy et al., 2005] E. D. Liddy, "The Book Chapter: Document Retrieval, Automatic",In Encyclopaedia of Language and Linguistics, 2nd Edition. Elsevier Press, 2005.

[Jarvelin and Kekalainen, 2000] Järvelin, K., & Kekäläinen, J. (2000, July). IR evaluation methods for retrieving highly relevant documents. In *Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval* (pp. 41-48). ACM.

[Sugiyama, Hatano, Yoshikawa and Uemura, 2004] Sugiyama, K., Hatano, K., Yoshikawa, M., & Uemura, S. (2004). User-Oriented Adaptive Web Information Retrieval Based on Implicit Observations. In *Advanced Web Technologies and Applications* (pp. 636-643). Springer Berlin Heidelberg.

[Costa and Roda, 2011] Costa, A., & Roda, F. (2011, May). Recommender systems by means of information retrieval. In *Proceedings of the International Conference on Web Intelligence, Mining and Semantics* (p. 57). ACM.

[Schafer, Frankowski, Herlocker, Sen, 2007] Schafer, J. B., Frankowski, D., Herlocker, J., & Sen, S. (2007). Collaborative filtering recommender systems. In *The adaptive web* (pp. 291-324). Springer Berlin Heidelberg.

[Fette, Sadeh, Tomasic, 2007] Fette, I., Sadeh, N., & Tomasic, A. (2007, May). Learning to detect phishing emails. In *Proceedings of the 16th international conference on World Wide Web* (pp. 649-656). ACM.

[Lv and Zhai, 2009] Lv, Y., & Zhai, C. (2009, July). Positional language models for information retrieval. In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval* (pp. 299-306). ACM.

[Ruthven and Lalmas, 2003] Ruthven, I., & Lalmas, M. (2003). A survey on the use of relevance feedback for information access systems. *The Knowledge Engineering Review*, *18*(02), 95-145.

[Goodrum et al., 2000] Goodrum, A. A. (2000). Image information retrieval: An overview of current research. *Informing Science*, *3*(2), 63-66.

[Lempel and Moran, 2003] Lempel, R., & Moran, S. (2003, May). Predictive caching and prefetching of query results in search engines. In *Proceedings of the 12th international conference on World Wide Web* (pp. 19-28). ACM.

[Acker, Roth and Bayer, 2008] Acker, R., Roth, C., & Bayer, R. (2008). Parallel query processing in databases on multicore architectures. In *Algorithms and Architectures for Parallel Processing* (pp. 2-13). Springer Berlin Heidelberg.

[Beame, Koutris and Suciu, 2013] Beame, P., Koutris, P., & Suciu, D. (2013, June). Communication steps for parallel query processing. In *Proceedings of the 32nd symposium on Principles of database systems* (pp. 273-284). ACM.

[Beame, Koutris, and Suciu, 2014] Beame, P., Koutris, P., & Suciu, D. (2014, June). Skew in parallel query processing. In *Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems* (pp. 212-223). ACM.

[Cahoon, McKinley and Lu, 2000] Cahoon, B., McKinley, K. S., & Lu, Z. (2000). Evaluating the performance of distributed architectures for information retrieval using a variety of workloads.*ACM Transactions on Information Systems (TOIS)*, *18*(1), 1-43.

[Choi, Kim, Ersoz, and Das] Choi, G. S., Kim, J. H., Ersoz, D., & Das, C. R. (2005, May). A multi-threaded PIPELINED Web server architecture for SMP/SoC machines. In*Proceedings of the 14th international conference on World Wide Web* (pp. 730-739). ACM.

[V. Pai, P. Druscheland W. Zwaenepoel] Pai, V. S., Druschel, P., & Zwaenepoel, W. (1999, June). Flash: An efficient and portable Web server. In *USENIX Annual Technical Conference, General Track* (pp. 199-212).

[Online]Zeus Web Server. Zeus Technology Limited. *http:// www.zeus.com,* 2003.

[Beniniand and Micheli] Benini, L., & De Micheli, G. (2002). Networks on chips: a new SoC paradigm.*Computer*, *35*(1), 70-78.

[Bell and Gray, 2003] Bell, G., & Gray, J. (2002). What's next in high-performance computing?.*Communications of the ACM*, *45*(2), 91-95.

[Edenfeld, Kahng, Rodgers and Zorian, 2004] Allan, A., Edenfeld, D., Joyner Jr, W. H., Kahng, A. B., Rodgers, M., & Zorian, Y. (2004). 2001 technology roadmap for semiconductors. *Computer*,*35*(1), 42-53.

[Dao, Xiang, Ling-Da, Long and Yang, 2005] Luan, X. D., Xie, Y. X., Wu, L. D., Mao, C. L., & Lao, S. Y. (2005, August). Information assistant: a novel initiative topic search engine. In *Machine Learning and Cybernetics, 2005. Proceedings of 2005 International Conference on* (Vol. 4, pp. 2363-2367). IEEE.

LIST OF TABLES